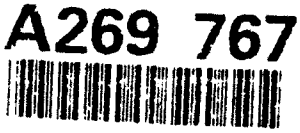
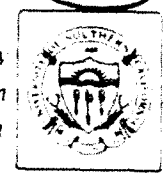
AD-A269 767



(12)



Investigating Production System Representations for Non-Combinatorial Match

Milind Tambe and Paul S. Rosenbloom
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

June 1993
ISI/RR-93-356

DTIC
ELECTE
SEP 22 1993
S E D

INFORMATION SCIENCES INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

STATEMENT A
Approved for public release

(12)

Investigating Production System Representations for Non-Combinatorial Match

Milind Tambe and Paul S. Rosenbloom
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

June 1993
ISI/RR-93-356

Accesion For
NTIS CRA&I [X]
DTIC TAB [ ]
Unannounced [ ]
Justification .....
By .....
Dist.ibution/
Availability Codes
Dist | Avail and/or Special
A-1

DTIC
ELECTE
SEP 22 1993
S E D

...TIC QUALITY INSPECTED 8

93-21827

...STATE...
...pproved for public rele...
Distribu...

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimated or any other aspect of this collection of information, including suggestings for reducing this burden to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1993	Research Report

4. TITLE AND SUBTITLE
Investigating Production System Representations for Non-Combinatorial Match

5. FUNDING NUMBERS
F33615-87-C-1499 and
N00039-86C-0033

6. AUTHOR(S)
Milind Tambe and Paul Rosenbloom

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-356

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Eliminating combinatorics from the match in production systems (or rule-based systems) is important for expert systems, real-time performance, machine learning (particularly with respect to the utility issue), parallel implementations and cognitive modeling. In [71], the unique-attributes engender a sufficiently negative set of trade-offs, so that investigating whether there are alternative representations that yield better trade-offs becomes of critical importance.

This article identifies two promising spaces of such alternatives, and explores a number of the alternatives within these spaces. The first is generated from local syntactic restrictions on working memory. Within this pace, unique-attributes is shown to be the best alternative possible. The second space comes from restrictions on the search performed during the match of individual productions (match-search). In particular, this space is derived from the combination of a new, more relaxed, match formulation (instantiationless match) and a set of restrictions derived from the constraint-satisfaction literature. Within this space, new alternatives are found that outperform unique-attributes in some, but not yet all domains

14. SUBJECT TERMS

15. NUMBER OF PAGES
47

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Investigating Production System Representations for Non-Combinatorial Match

Milind Tambe
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
tambe@cs.cmu.edu

Paul S. Rosenbloom
Information Sciences Institute &
Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
rosenbloom@isi.edu

Abstract

Eliminating combinatorics from the match in production systems (or rule-based systems) is important for expert systems, real-time performance, machine learning (particularly with respect to the utility issue), parallel implementations and cognitive modeling. In [71], the unique-attribute representation was introduced to eliminate combinatorics from the match. However, in so doing, unique-attributes engender a sufficiently negative set of trade-offs, so that investigating whether there are alternative representations that yield better trade-offs becomes of critical importance.

This article identifies two promising spaces of such alternatives, and explores a number of the alternatives within these spaces. The first space is generated from local syntactic restrictions on working memory. Within this space, unique-attributes is shown to be the best alternative possible. The second space comes from restrictions on the search performed during the match of individual productions (match-search). In particular, this space is derived from the combination of a new, more relaxed, match formulation (instantiationless match) and a set of restrictions derived from the constraint-satisfaction literature. Within this space, new alternatives are found that outperform unique-attributes in some, but not yet all, domains.

1. Introduction

Production systems (or rule-based systems) are used extensively in investigating and building a wide range of intelligent systems; including expert systems [47, 77], real-time systems [38], learning systems [51, 40], parallel systems [30, 52], and cognitive models [5, 59, 78]. However, despite this extensive use, there remains a key performance bottleneck within the core recognize-act cycle that drives production-system behavior. The problem turns out to not be with action — once the system recognizes which production(s) should fire, the act phase, and the working-memory modifications that it engenders, require minimal resources. Instead, the critical performance bottleneck turns out to be with the recognition process that enables this action. During the recognize phase, productions are matched against the data elements in working memory via a combinatoric (NP-hard) process. (Section 2 of this article provides additional background on production systems and match.)

This combinatoric match process creates problems in all of the areas in which production systems have been applied. In expert systems, it can cause performance degradations that impose a significant burden on the programmer to restructure the system so as to eliminate the causes of the degradation (see, for example, Section 6.2 of [15]). In real-time systems based on production systems, achieving guaranteed (or predictable) response times is critical [10, 22, 38]. However, with a combinatorial match, many real-time tasks are precluded because only unacceptably large exponential bounds can be guaranteed for the match. In systems that learn new productions, combinatorial match can lead to a slowdown with learning — causing what is known as the *utility problem* in explanation-based learning (EBL) [49] — rather than the anticipated speedup [49, 75]. These expensive productions can, in the extreme, convert a subexponential problem-solving effort into an exponential match problem, causing a severe degradation in performance. In parallel systems, large variances can occur in matching individual productions and even in matching individual conditions of productions [30]. These variances result in load-balancing problems, which penalize the parallel implementation with synchronization and communication overheads [31, 74]. In cognitive models, the problem is plausibility (in addition to functionality). As pointed out in [59], it is possible to encode a traveling salesman problem into the match of a single production, and *"there are good reasons to believe that humans do not have this much power in the recognition match, so that it ought to be replaced by a weaker capability"* (p. 253).

A variety of types of solutions to the problem of combinatoric match have been proposed within the individual problem areas. In expert systems, it is suggested that the programmer hand modify the rules to reduce the sources of match cost [15]. In real-time systems, proposed approaches to improving predictability include restricting variability through the setting of a priori limits on the amount of work that can occur during the match (such as through a preset limit on the amount of effort spent matching individual production conditions) [32]; gathering run-time statistics on the match process [10]; and analyzing at compile time all possible ways that the productions can match to working memory [76]. In learning, the utility problem has been addressed by increasing the selectivity of production acquisition, deletion, and/or use [23, 35, 46, 48]; and by modifying learned productions [17, 49]. In parallel systems, the load-balancing problem has been alleviated by using a fine-grained decomposition of the match computation in parallelization [30].

In our own work, we are attempting to take an integrated approach to the problem of combinatoric match, with the intent of ultimately developing a solution that will prove adequate across all of these problem areas. We have been using Soar, an integrated problem-solving and learning system that has been applied in all of the above problem areas, in driving this effort. Soar has already been well-reported in the literature [40, 66]. For the purposes of this article, we will need to focus on just two key aspects of Soar: its use of a production system to implement its knowledge-base; and its use of chunking [41], a variant of EBL [21, 57, 65], to learn new productions.

In [71, 75], we proposed one integrated approach to eliminating combinatoric match based on restricting the representation used in the production system. In particular, we proposed and evaluated a specific representational restriction, called *unique-attributes*, that guarantees a linear bound (in the number of production conditions) on the match cost of individual productions. This simple restriction reduces the load on the programmer by eliminating the need to hand modify expensive rules, greatly enhances real-time predictability (while hopefully reducing load-balancing problems in parallel systems) by reducing the variability in match time, effectively eliminates the cost of learned rules as a factor in the utility problem,[1] and eliminates the plausibility problem (at least this particular one) dogging cognitive models. Because of such advantages, unique-attributes have now been used in encoding a variety of complex tasks [4, 11, 12, 42, 43, 68, 70]. Outside of Soar, Chalasani and Altmann [1989] have pointed out that the knowledge representation scheme adopted by Theo, a frame-based architecture for problem solving and learning [56], already corresponds to the unique-attribute representation. Unique-attributes have also been mapped over (in an extended form) to the PAMELA expert system [6, 9], to the **K** production system [14], and simple unique-attribute set structures have been mapped over to Prolog control rules [18].

Despite all of this promise, the unique-attribute representation does not come without its own set of costs: it imposes restrictions on expressiveness. In particular, it imposes restrictions on the processing of sets of objects in working memory: a single condition in a production can no longer examine (or consider) more than one element of a set at a time. Such processing of multiple set elements, to the extent that it is really needed, must now occur across multiple production matches and firings. This reduction in expressiveness of individual rules can yield both reduced generality in learned rules and additional programmer effort in encoding tasks [75]. Thus, though unique-attributes demonstrated the overall promise of eliminating combinatoric match by restricting expressiveness, it was not at all clear by the end of the investigation that unique-attributes provided the best such restriction. (Section 3 of this article reviews what is presently known about unique-attributes.)

The purpose of the present article is to take a first step in addressing the question of whether there exist representational restrictions other than unique-attributes that can eliminate match combinatorics while introducing fewer negative trade-offs[2]. The ideal way to prosecute such an investigation would be to first characterize the space of all possible alternatives, and then to systematically evaluate them. We have not

---

[1] Two other recent attacks on the utility problem based on imposing restrictions can be found in [79] and [23].

[2] This article is an expanded version of an earlier conference paper [72].

yet been able to derive such a single space of alternatives that is guaranteed to be complete. However, we have been able to characterize two distinct spaces that between them do cover a significant number of interesting alternatives. One interesting question is then how to evaluate these alternatives. Section 4 introduces the approach that has been developed for evaluating the alternatives. It combines a set of absolute requirements (i.e., constraints), any of whose violation is sufficient to eliminate an alternative from consideration, with a set of relative requirements that can be used to compare alternatives that meet the absolute requirements.

Our two spaces of alternatives are actually derived from two of the absolute constraints used in evaluating the alternatives (whether any of the other constraints can also be so used remains an open question). The first constraint is that any representational restriction must fit naturally as a local syntactic restriction on the production system. The first space of alternatives is thus comprised of representations that all provide local syntactic restrictions on the production system (but may or may not a priori provide polynomial match bounds). For example, for unique-attributes, the local syntactic restriction is that the set of attribute-value pairs which make up an object in working memory can contain at most one value for any particular attribute; that is, each attribute of each object has a unique value. In fact, the name "unique-attributes" was derived from this space — each attribute is unique in value. To go beyond unique-attributes we view working memory as a labeled, directed graph structure, in which each attribute-value pair of an object specifies a labeled, directed edge from an "object" node to a "value" node. We then systematically generate a set of restrictions on the edges entering and leaving these nodes.

Because alternatives generated in this first space are not a priori guaranteed to provide polynomial match bounds, they must be explicitly tested with respect to their complexity. For unique-attributes, the linear match bound easily satisfies this requirement. Whether or not any of the other elements of this space will also do so, while also yielding better overall trade-offs than do unique-attributes, is the first major issue addressed by this paper — in Section 5. The rather surprising conclusion from this investigation is that unique-attributes are actually the best representation possible within the space of local graph restrictions that has been identified.

The second constraint is that the representational restriction must yield a polynomial bound on the search performed during the match of individual productions (match-search). The second space of alternatives thus consists of polynomially bounded match-search topologies. The unique-attributes restriction provides a linear search topology. Each node in the search has at most one successor (and the length of the search is bounded by the number of conditions in a production). Other polynomial search structures are conceivable, such as a fixed overall depth limit on the search, but it is not clear a priori either whether they yield sensible syntactic restrictions or whether they provide a better set of trade-offs than does unique-attributes. The second major issue addressed by this paper is whether such alternatives do exist. The focus here will be on a novel set of such alternatives that has been generated by weakening the requirements on what the match must produce. Traditional production match algorithms produce *instantiations*; that is, compatible combinations of bindings of condition variables. If there are m conditions and n bindings for the variables in each condition, and all of the bindings for each condition are compatible with all of the bindings for the other conditions, then n^m instantiations must be produced

by the match. However, with the new *instantiationless match* formulation that is presented in Section 6, all that must be provided by the match is the m sets of n items that are compatible (that is, $n*m$ items).

While instantiationless match can't by itself guarantee polynomial match-search, it can when combined with other restrictions. Fortunately, there turns out to be a mapping between instantiationless match and constraint-satisfaction algorithms [44, 20, 26] that provides a rich body of complexity results on different representational restrictions, and which can therefore act as a generator for the space of polynomially bounded instantiationless-match alternatives. Most of the results reported here, in Sections 7 and 8, are based on combining instantiationless match with one simple such restriction: each production's conditions must be structured as a tree. This *instantiationless-tree* formulation is only the simplest of a range of intriguing possibilities; however, it is sufficient to show both the potential and the problems with approaches based on instantiationless match. In Section 9 we go a bit beyond the instantiationless-tree formulation, to look at the use of partial-2-trees [20], in order to illustrate what can be gained in moving up to one of the more complex possibilities.

Section 10 summarizes the ideas developed here and relates them to other efficiency issues in productions systems and constraint satisfaction.

2. Production Systems

Section 2.1 explains the terminology used in the remainder of this article. Section 2.2 presents a *model production system* that abstracts away from some of the idiosyncratic details of Soar, to both simplify the overall analysis and allow transfer of these results to other systems. Section 2.3 provides a simple model of the production match, to free the analysis presented in this article from the complexities of the implementation.

2.1. Terminology

Production systems are composed of two parts: production memory and working memory. Production memory is composed of a set of productions (or condition-action rules). The condition side of a production is called the left-hand side (or LHS), while the action side is called the right-hand side (or RHS). The representation in the productions and working memory is usually based on objects that are described in terms of attribute-value pairs. Figure 1 shows an example production system. The production memory consists of a single production (Figure 1-a), which contains four conditions (condition elements or CEs) and one action. In the figure, up-arrows (^) indicate attribute names and angle brackets (<>) indicate variables.

Working memory is composed of a set of data items called working memory elements (or WMEs). Figure 1-b shows the ten WMEs in the example production system. These WMEs describe objects A,B etc. that are described in terms of their connectedness to other objects. While the production's conditions contain variables (<x>, <y>, etc.) or constants (`current-position`, etc.), the WMEs can only contain constants (A, B, etc.). In Figure 1-a, the production's conditions test for a path of length three between the current position (<x>) and some other point (<w>).

```
(Production::Length
    (<x> ^is current-position)
    (<x> ^connected <y>)
    (<y> ^connected <z>)
    (<z> ^connected <w>)
-->
    (write path of length 3 from <x> to <w> ))
```

(a)

```
(A ^is current-position)
(A ^connected B)
(A ^connected C)
(B ^connected D)
(C ^connected D)
(D ^connected E)
(D ^connected F)
```

(b)

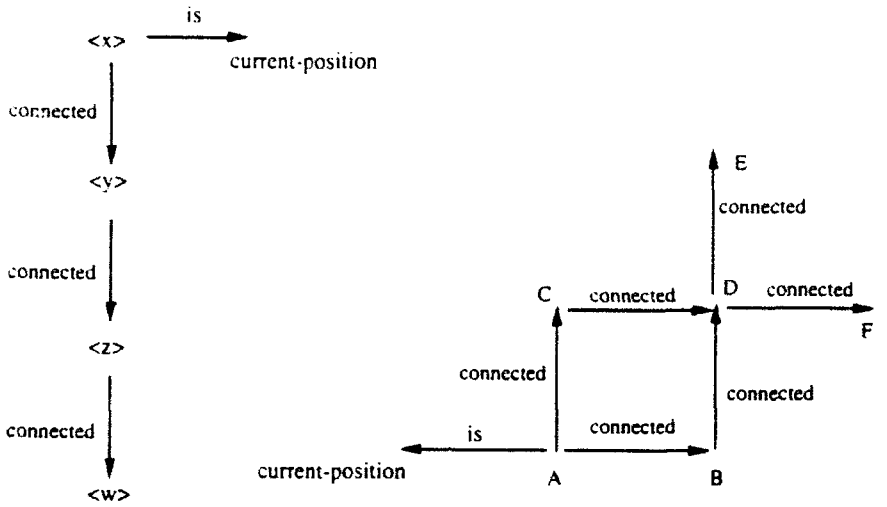
Figure 1: An example production system: (a) a production, (b) working memory.

Currently, production systems use a type of match that involves the computation of one or more *instantiations* of the production with the working memory. This form of match will henceforth be referred to as *instantiation match*.[3] As mentioned in the introduction, an instantiation is a collection of consistent bindings for the variables in the production, resulting from the match with the working memory. In our example, there are two paths of length three between the current position A and point E, and two paths of length three between A and point F. This generates four instantiations for the production: (<x> = A, <y> = B, <z> = D, <w> = E), (<x> = A, <y> = B, <z> = D, <w> = F), (<x> = A, <y> = C, <z> = D, <w> = E), and (<x> = A, <y> = C, <z> = D, <w> = F). When one of these instantiations fires, the action side of the production is executed in the context of the variable bindings for that instantiation. For instance, if (<x> = A, <y> = B, <z> = D, <w> = E) fires, the action side of the production in Figure 1 is executed with the binding A for variable <x> and binding E for <y>. This results in the fact (path of length 3 from A to E) being written.

The complexity of the production-match process can be understood by using a directed labeled graph representation for productions and working memory. Consider the production shown in Figure 1-a. This production encodes a directed labeled graph, where the variables and constants in its conditions are nodes of the graph and attributes are the edges between the nodes (Figure 2-a). Similarly, the WMEs in Figure 1-b encode another graph (Figure 2-b). Here, symbols in working memory are nodes of the graph and the attributes are again the edges between the nodes. In general, productions and WMEs can encode arbitrary directed graphs. The task of generating a single instantiation then reduces to the problem of subgraph isomorphism — finding a subgraph of the working memory graph that is isomorphic to the production

[3]In previous work, this form of match was referred to as *token match* [72]; however, the name has been changed to emphasize what the match produces rather than how it produces it. Likewise the earlier usage of *tokenless match* has been replaced here by *instantiationless match*.

graph — which is a well-known NP-complete problem [27, 49]. The problem becomes even harder if more than one instantiation is to be computed.



(a) (b)

Figure 2: An example production system: (a) graph of the production from Figure 1-a, (b) graph of the working memory from Figure 1-b.

2.2. The Model Production System

To abstract away from some of the idiosyncratic details of Soar, a *model production system* language has been developed for use in this investigation. This model is a close enough approximation to Soar's production system that analysis based on it applies almost directly to Soar. Yet, it is also abstract enough to simplify the overall analysis and to allow transfer of the analysis other systems. In the remainder of this article, the model production system is used for any theoretical analysis, and Soar is used for experimental investigation.

The model production system is based on the following assumptions:

1. *Computation of all instantiations*: When a production matches working memory, all possible instantiations of the production with the working memory are computed. Besides Soar, some other production systems depend on this assumption as well, including Prodigy [50] (in matching its search control rules), and some set-oriented production systems from the field of expert-database systems [29, 33]. Nearly all implementations of OPS5 [15] and related systems also make this assumption.[4]

[4]It has recently been pointed out that OPS5 need not compute all instantiations — instead, computation of just the dominant instantiation can be sufficient [55]. However, there are still cases in which the direct computation of the dominant instantiation is sufficiently complex that it is preferable to compute all of the instantiations first and then select the dominant one rather than trying to compute it directly [6, 7].

2. *Triple-based representation*: The production system representation is based on (object ^attribute value) triples. For instance, WMEs are of the form `(A ^connected B)`, while conditions of productions are of the form `(<x> ^connected <y>)`. In the conditions, the object field contains variables, the attribute field contains constants, and the value field may contain variables or constants. Note that it is possible to have different predicate tests on the value field, e.g., `(<a> ^length <>5)`, etc. Triple-based representations, and their close relatives (such as binary relations), are quite common throughout AI.

3. *Pre-bound object field*: During the match, a variable in the object field of a condition is assumed to be pre-bound. That is, a variable in the object field must occur in some previous condition of the production. Thus, unbound variables can only appear in the value fields of conditions. (Obviously, the first condition in a production cannot adhere to this constraint.) For instance, consider the production in Figure 1-a. The variables in the object fields of the second, third and fourth conditions are pre-bound. Unbound variables like <y> in the second condition can only occur in the value field.

   This constraint corresponds to the connectivity heuristic, which is used in ordering conditions of productions to achieve better match efficiency [37, 67]. An identical restriction of a pre-bound object field is also used in the ALL knowledge representation language [19]. In terms of the graph representation, this constraint implies that the variable occurring in the identifier field of the first condition acts as a root (or originating point) of the production graph. For instance, in Figure 1-a, the variable <x> acts as the root of the production graph.

4. *Single match for the first condition*: The first condition of a production can only match a single WME. For instance, the first condition in the production in Figure 1-a can only match the WME `(A ^is current-position)`.

   While the first three assumptions play a significant role in our exploration of a polynomial match bound, this fourth assumption only simplifies our analysis. Since the first condition in a production cannot adhere to the constraint of a pre-bound object field, this constraint limits the match for the first condition. Eliminating this constraint may only lead to a slightly higher polynomial bound, e.g., a linear term may become a square term.

In addition to Soar, other systems that meet the constraints of this abstract model include the K production system [14, 13] — where unique-attributes have in fact already been applied — and schemas in the ART production system [36, 3]. However, it is important to note that our results are not strictly limited in their applicability to systems that conform to the underlying assumptions. For other systems that don't completely map into this model, it may be feasible to restrict them to do so. For example, Soar's production system actually arose as a restricted version of OPS5 [39]. Thus, it will be possible to restrict OPS5 to conform to this model. Beyond this, it may also be possible to further abstract this model so that the main results can be carried over directly without further restriction. For example, Barachini and Veretenual [9] have mapped unique-attributes on to the PAMELA expert system [8]. Their mapping works in a domain where WMEs primarily contain integers as values. It exploits the integer values by using pre-determined intervals (e.g., -5..5) to partition the set of WMEs matching a condition into smaller subsets. If each subset contains exactly one WME, then a representation similar to unique-attributes emerges. In general, the subsets contain more than a single WME, and a more generalized form of unique-attributes emerges. While this scheme does not guarantee a polynomial bound on the match, Barachini and Veretenual show that in practice it provides a good upper bound on the match for individual productions.

In addition to the base assumptions about production systems, the model production system also assumes that, as in Soar, a knowledge compilation mechanism such as explanation-based learning/chunking [21, 57, 65], is used to learn new rules. Mechanisms of this type compile new rules from the performance traces of existing (old) rules, with the intent of speeding up future performance.

2.3. Modeling Match Algorithms: The K-search Model

The k-search model of production match covers match algorithms that find all possible solutions, without the aid of heuristics. This includes widely used algorithms such as Rete [25] and its variations [6], Treat [52], as well as others, such as Tree [13], Scaffolding [62], Dynamic-join [60], etc. The k-search model allows analysis of match cost independent of the complexities of the physical machine or match algorithms typically used in production systems. The model is based on the notion of tokens. Tokens are partial instantiations of productions indicating what conditions have matched and under what variable bindings.

Consider the match of the production in Figure 1-a with the working memory in Figure 1-b. While matching the production, tokens will be generated, some of which are: `(2; <x> = A, <y> = B)`, `(2; <x> = A, <y> = C)` etc. The first number in the token indicates the number of conditions matched and the other elements indicate the bindings for the variables. Thus, the token `(2; <x> = A, <y> = B)` shows that the first two condition elements were matched with the bindings `A` for variable `<x>`, and `B` for variable `<y>`. An *instantiation* is then just any token generated as the result of matching the last condition of a production. Each such instantiation encodes one set of consistent bindings for all of the variables in a production. All of the algorithms cited above generate instantiations as the end result of the match, so they can be referred to as *instantiation-match* algorithms.

The tokens generated in the match can be represented in the form of a tree, as shown in Figure 3. Each arc in this tree represents a token. The tree represents the search conducted by the matcher in order to match the production. Since this search is done *within* the productions, i.e., in service of retrieving information from the system's *knowledge* base, it is called *k-search* in order to distinguish it from the higher level problem solving search that may also be occurring across multiple production firings

Measurements indicate that the time spent in match per token is approximately constant [74, 58, 49]. Therefore, we assume the *number of tokens* in the k-search tree to be a reasonable estimate of the work done in performing match. In particular, this simple estimate appears to be sufficient for comparing combinatorial and non-combinatorial matches for productions[5].

The source of combinatorics in production match can now be understood. Combinatorics arises in production match due to the branching in the k-search tree. This branching arises due to multiple WMEs that match a single condition of the production. For instance, in Figure 3, the two WMEs `(A ^connected B)` or `(A ^connected C)` lead to branching in the k-search tree while matching the

[5]Tokens may not have equal cost across match algorithms [58]. However, the cost difference is a small constant and can be ignored for the purpose of comparing combinatorial and non-combinatorial match.

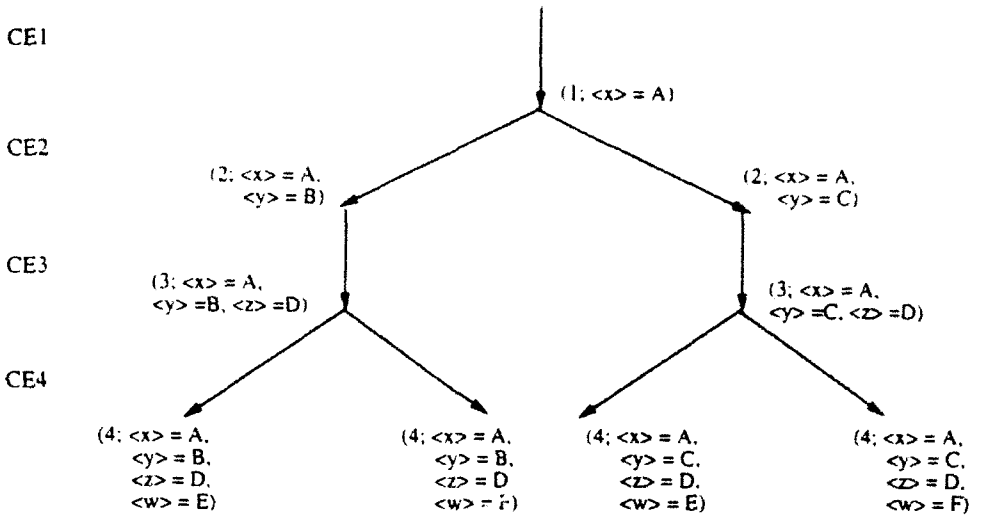


Figure 3: The k-search tree of tokens generated when the production in Figure 1-a matches the working memory in Figure 1-b.

second condition. The WMEs matching the fourth condition lead to further branching in the k-search tree. In a production with multiple conditions, a cascading effect of such branching leads to an exponential match effort, i.e., a k-search tree with a number of tokens $O(\#WMEs^{\#conditions})$ [75]. This cascading effect is also referred to as the *cross-product* effect [30, 31], since it yields the cross-product of the bindings for the variables[6].

Given the representational constraints in the model production system, the WMEs that lead to branching in the k-search tree can be identified clearly. In particular, in the model production system, an unbound variable can only occur in the value fields of conditions. Therefore, branching can only occur in matching a condition of the model production system if there are multiple possible bindings for the value field. This situation can only occur in the presence of a *multi-attribute*, i.e., a set of WMEs with identical object and attribute fields, but with differences in the value field. For example, the following two WMEs from Figure 1-b form a multi-attribute: `(A ^connected B)`, `(A ^connected C)`. Here, `connected` is a multi-attribute of `A`, with values `B`, and `C`. The WMEs `(D ^connected E)`, `(D ^connected F)` form the second multi-attribute. Matching such multi-attributes leads to branching in the k-search tree, as shown in Figure 3.

[6]Two recent efforts have shown how this cross-product effect can be reduced by only computing a single instantiation [55] or by using a collection-oriented match (based on the work here on instantiationless match) [1]; however, neither approach can completely eliminate cross products and their combinatorics.

3. Unique-attributes

The principle behind the unique-attribute representation is to eliminate the branching in the k-search tree. With unique-attributes, a condition can match only a single WME. This limits the number of tokens in the k-search tree to the number of conditions in the production. Thus, *the match cost of a production becomes linear in the number of conditions*, i.e., the match cost is bounded linearly rather than being NP-hard. For productions containing variables, this O(conditions) match bound is optimal, since all the conditions of a production must be examined for match in any event.

For the model production system, eliminating branching implies the elimination of multi-attributes from working memory (hence the name of the *unique-attribute* representation). Figure 4 shows the WMEs from Figure 1 in the unique-attribute representation. Note that the production system without any representational restrictions will now be referred to as the *unrestricted* representation.

(A ^is current-position)

(A ^right B)

(A ^up C)

(B ^up D)

(C ^right D)

(D ^up E)

(D ^right F)

Figure 4: An example of working memory with the unique-attribute representation.

The important step involved with the unique-attribute representation is the analysis of: its impact on performance; how it interacts with learning; what its tradeoffs are; whether it can support performance of large tasks; and so on. These issues are addressed in detail in [75, 70]. However, in order to illustrate some of the answers, the following section reproduces the analysis of a simple task, the Grid task, from [75]. This task is related to the example presented in Figure 1. Besides illustrating the issues in unique-attributes, this analysis is also useful background for the discussion of the Grid task in instantiationless match presented in Section 8.1.

3.1. The Grid Task

The Grid task is shown in Figure 5. To simplify some of the following analysis, assume that the grid is infinite. The problem is to go from point A to point B, a path of length four. This problem is solved first using the unrestricted representation and then using unique-attributes.

In the unrestricted version, the grid is represented using *connected* as a multi-attribute of a point on the grid. Any point Y adjacent to a point X on the grid is represented as: `(X ^connected Y)`. The state of the problem solver is its position on the Grid, which is encoded as: `(A ^is current-state)`. There is only a single operator: *move*. If the current position is at point *x*, then for each point *y* connected

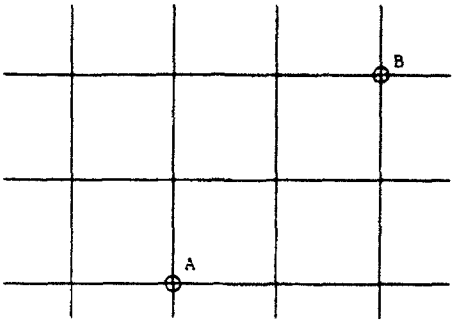


Figure 5: The Grid task.

to point x, the operator *move* will be instantiated. The problem-solver will solve the problem perhaps using some heuristics, or guidance from an outside teacher [28], generating a k-search tree of tokens as shown in Figure 6-a. This process generates 16 tokens, with four tokens per step generated from the four options available to the problem solver at each point on the grid. The learned rule formed after solving the task is shown in Figure 6-b. The rule says that if the goal is to reach a point <d>, and if the current position is point <x>, and if there is a path of length four between them, then prefer the instantiated *move* operator along that path. (The prefer action indicates a preference for a path from <x> to <y> over all other paths.) This rule does not consider the points along which the path goes or the direction the path takes. The rule will therefore transfer to all pairs of points with a path length of four between them.

Figure 6-c shows the k-search tree formed in matching the rule. For the sake of simplicity in the analysis, the first and last conditions on the rule are not considered. (Considering these conditions does not change the results in any significant way.) Here, each condition has multiplied the number of tokens by four, which is the number of points connected to any given point. Since there are four conditions in the rule (for a path of length four), the total number of tokens is $(4^1 + 4^2 + 4^3 + 4^4)$. Thus, for a path length of p, the cost will be $\sum_{k=1}^{p} 4^k$.

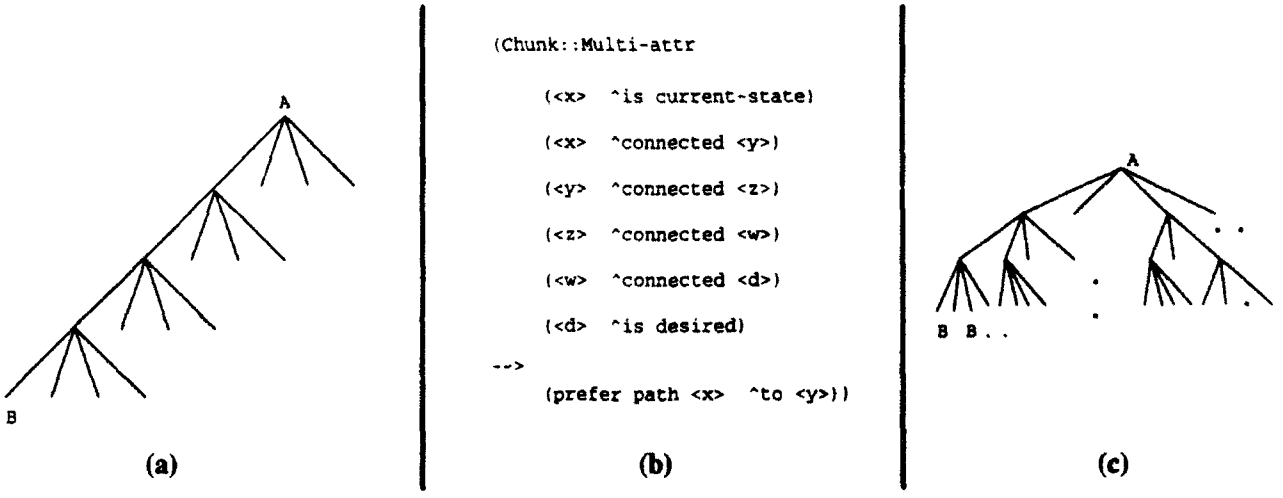


Figure 6: The Grid task in the unrestricted representation: (a) the k-search tree for the entire task consisting of four problem solving steps; (b) the learned rule (the prefer action indicates a preference for a path from <x> to <y> over all other paths); (c) the k-search tree formed in matching the learned rule.

In the unique-attribute version, each location points to its four adjacent locations using specific unique-attributes: *up*, *down*, *left*, and *right*. Instead of one *move* operator, there are four different operators, *move-up*, *move-left*, *move-right*, and *move-down*. The problem solver again moves from A to B exactly as in 6-a, generating an identical k-search tree of 16 tokens as shown in Figure 7-a. However, the learned rule formed in this process (Figure 7-b) is different. It says that if the goal is to move to point <d> from point <x> and if the connection between the two points is through the specific relation (*up-right-up-right*) described, then move up one step. The k-search tree formed is shown in Figure 7-c. There are only four tokens per step formed in this case. The learned rule is much cheaper than the rule in the unrestricted case. However, the rule will transfer only if the two points are connected in a specific manner — *up-right-up-right* in this case, as opposed to any arbitrary connection of length four in the earlier case.

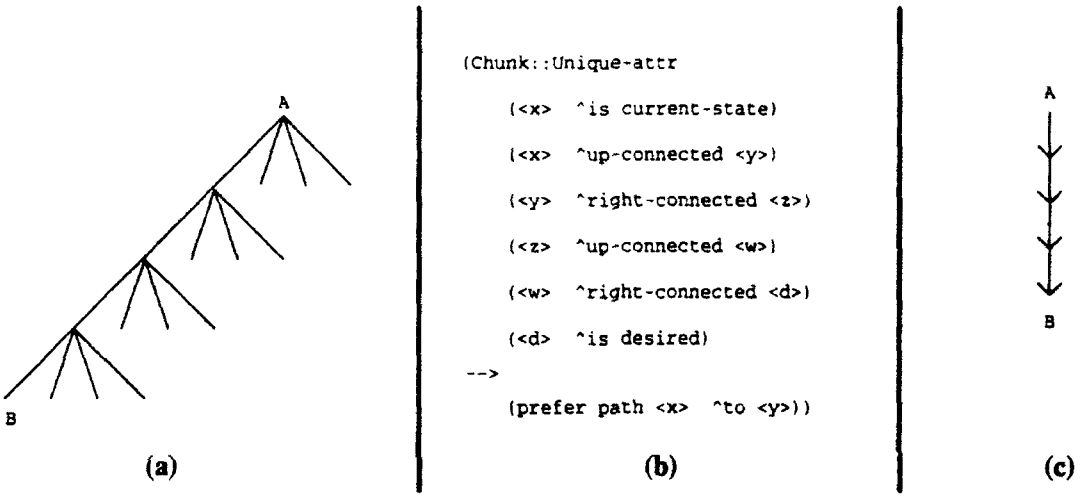


Figure 7: The Grid task with unique-attributes: (a) the k-search tree for the entire task consisting of four problem solving steps; (b) the chunk formed (the prefer action indicates a preference for a path from <x> to <y> over all other paths); and (c) the k-search tree formed in matching the learned rule.

Table I summarizes the cost and generality of the two representations. The match cost is measured in tokens. The generality is measured in terms of the number of transfers in an *nxn* grid, i.e., the number of source-destination pairs to which the learned rule can transfer (or apply). The length of the path traversed in the grid is assumed to be *p*.

Representation used	Match cost per rule	Generality in number of transfers	Number of learned rules for same level of generality	Match cost with all rules for same level of generality
Unrestricted	$(4^{p+1} - 4)/3$	$n^2*(p+1)^2$	1	$(4^{p+1} - 4)/3$
Unique-attributes	p	n^2	$(p+1)^2$	$(p+1)^2*p$

Table I: The cost and generality of the unrestricted and unique-attribute representations for the Grid task. Here *n* is the number of nodes in the grid and *p* is the path length.

This example illustrates some of the tradeoffs with unique-attributes. Consider the total match cost of the two systems after they have both learned enough rules to cover all paths of length p (this requires only one rule for the unrestricted representation, but $(p + 1)^2$ rules for unique-attributes). As expected, the cost

of the match in the unrestricted representation is exponential in the path length p. However, the cost of match with unique-attributes is a small polynomial. This reduction in match cost, despite the equivalent level of generality, comes about because the unrestricted system computes all possible paths of length p from the start point to the destination point. However, the action side only requires information about the first step in a path that leads to the destination point. If two paths share the first step, then only one of them should really be computed. Thus, much of the computation in the unrestricted system is unnecessary or extra. The unique-attribute system avoids this unnecessary k-search. A single rule in unique-attributes computes one path to one destination; and different rules cover the paths to different destinations. The unique-attribute system is thus able to reduce its k-search to a small polynomial.

In many situations (especially those involving expensive learned rules), the unrestricted representation causes a large amount of such unnecessary k-search. By getting rid of this unnecessary k-search, the unique-attribute system can provide a big performance improvement. Additional performance improvements can also be obtained by using the more efficient implementation technology allowed by the unique-attribute representation, e.g., for Soar systems, the Uni-Rete matcher in unique-attributes can provide a more than 10-fold speedup over the Rete matcher employed in the unrestricted representation [73]. In some situations, the unrestricted representation does not engage in unnecessary k-search. In such situations, the performance gains in unique-attributes are only in terms of the more efficient implementation technology.

The Grid task also illustrates the price paid by unique-attributes, in terms of loss of expressive power, for obtaining this performance improvement. In the unrestricted system, multi-attributes allow a single condition in a production to successfully match an entire set of WMEs. For instance, a single condition can successfully match the multi-attribute: `(A ^connected B)` and `(A ^connected C)`. Here, the multi-attribute *connected* essentially represents an unstructured set of objects `{B, C}` that are connected to point `A`. However, with unique-attributes, all such sets have to be represented so that a single condition can successfully match only a single WME. This can be achieved by structuring the set of objects. In Figure 4, the set of objects `B` and `C` is structured in a task-specific manner by introducing a different attribute for each object. However, more task-independent structures, such as lists and trees, are also possible. For instance, `(A ^first-connected B) (B ^next C) (C ^next nil)` is a list structure for the two objects `B` and `C`. Since the type of set structure employed can impact problem-solving performance, introducing such structured sets appropriately is not always easy. For a further discussion on the issues involved in structuring sets, see [12, 70].

Finally, the Grid task illustrates the loss in coverage or generality of productions engendered by unique-attributes. That is, a larger number of unique-attribute productions may be required to gain the same coverage as a single unrestricted production. In the Grid task, $(p + 1)^2$ more rules have to be learned with the unique-attribute representation. This loss of generality results from the fact that, with unique-attributes, each production engages in a lower amount of k-search. To compensate for this weakened k-search, a larger number of unique-attribute productions may be required. An alternative way of compensating is to actually perform this search across multiple recognize-act cycles. Consider a multi-attribute set that is replaced by a list-structured unique-attribute, as illustrated in the previous paragraph.

The system may actually process this list in multiple recognize-act cycles, by looping over the list, and examining one element per cycle. The advantages and disadvantages of these two approaches to compensating for weakened k-search, and their interaction with learning are discussed in detail in [75]. The key point to note here is that these techniques result in additional encoding effort in some tasks. This encoding difficulty, along with the concomitant loss in generality, provides the motivation to search for alternative representations.

4. Evaluating Alternative Approaches

An important question arises as we search for alternatives to the unique-attribute representation: How should we select the best among these alternative representation schemes? Ideally, a representation scheme that imposes the least restriction on expressiveness and engages in the best tradeoff would be the best one. However, as shown in this article, different representation schemes restrict the production system language in different ways and engage in qualitatively different tradeoffs. Given these diverse tradeoffs, the concepts of the least restrictive approach and the best tradeoff are not precisely defined.

Thus, a complex scheme of evaluation becomes necessary to select the best among the different representation schemes. We have devised a method that evaluates the schemes on two levels: absolute and relative. The *absolute level* evaluates if a representation scheme meets some minimum requirements of acceptability. Any scheme must meet these requirements. The *relative level* compares different acceptable schemes to select the better one.

Evaluation at the absolute level is performed using a set of *absolute requirements*. These requirements are described below:

1. *Polynomial bound on match complexity:* Any given representation scheme must achieve a polynomial bound on match complexity. This requirement is primarily motivated by the issue of real-time performance, from Section 1. However, this bound is also expected to alleviate our concern regarding parallel implementations and the modeling of human cognition.

2. *Closure under learning*: If the productions and the working memory meet the expressiveness restrictions imposed by a given scheme before learning, the learned productions should also meet the restrictions. In Soar, this requirement implies that chunks should not violate the expressiveness restrictions of a given approach. If chunking violates the restrictions and chunks require a combinatorial match, clearly that defeats the purpose of this exercise. (Small modifications to the learning mechanism to meet this requirement are admissible.) In addition to the issues that motivated the first absolute requirement, the key issue motivating this requirement is the utility issue in machine learning.

3. *Correctness of match:* Any given method of restricting expressiveness must not cause match to provide incorrect results.

4. *Expressible as a local syntactic restriction:* For a restriction to be usable by programmers, it must be obvious when s/he is about to violate it; for example, the unique-attributes restriction is violated whenever an attribute in working memory has more than one value. If a violation can only be detected via some complex algorithm, the restriction won't be operational.

The relative level of evaluation compares the different approaches for restricting expressiveness using

the following *relative requirements*:

1. *Relative expressive adequacy:* This term refers to the investigations of how easy/difficult it is to encode various tasks in a given representation. Schemes that allow easier encoding are preferred. This is an informal requirement: it is analogous to evaluating the ease of programming in different programming languages.

2. *Relative Efficiency:* Within the space of representations with a polynomially bounded match, schemes with smaller polynomial bounds are preferred.

3. *Learning generality:* This requirement is based on the notion of the number of rules required to cover a space of problems. Consider two different situations. In the first, suppose a small number (N_1) of productions are learned for covering a space of problems. In the second, suppose N_2 number of productions (greater than N_1) have to be learned to cover the same space of problems. Then the productions learned in the first situation are more general than the productions learned in the second situation. Schemes that provide higher learning generality are preferred.

4. *Completeness:* If learned rules only affect the speed of performance, there is no significant problem with not learning at some occasions when learning is possible. However, if learned rules can also affect correctness — that is, they represent new knowledge — failing to learn a rule that could have been learned will degrade more than just efficiency. Thus, schemes that learn whenever new information is available are preferable to those that are more selective.

5. *Uniformity:* Representations that do not introduce arbitrary divisions in productions or working memory are preferred. For instance, a representation scheme that explicitly labels and separates multi-attribute WMEs from unique-attribute WMEs violates the principle of uniformity.

The uniformity requirement prefers simplicity in the representation scheme. Though this is clearly important when building integrated architectures like Soar, it still has a lower priority than the other relative requirements, and so may be sacrificed in service of them. Beyond this, it is difficult to prioritize the other relative requirements. This leads to difficulties in comparing different representation schemes in some situations. For instance, how can two representation schemes be compared when one has better efficiency but lower learning generality than the other? As seen later, despite such difficulties, the requirements outlined above are a substantial help in comparing different representation schemes.

As an example of the use of these requirements, consider a combination of unique- and multi-attributes that attempts to introduce limited amounts of multi-attributes to gain expressibility without sacrificing efficiency. First, suppose this representation adheres to the principle of uniformity and does not explicitly separate out unique- and multi-attributes from each other. Then, it has no way of controlling the number of multi-attributes matching a production — it is exactly like the unrestricted system. The match cost of productions becomes unpredictable and the polynomial match bound requirement is violated.

Now, suppose the violation of the principle of uniformity is accepted: the system explicitly labels and separates unique- and multi-attributes. It can then bound the number of multi-attribute-matching conditions in any single production — thus controlling the match cost. However, it is possible for such a system to learn rules that are expensive to match, i.e., to learn rules where the number of multi-attribute-matching conditions exceeds the specified bound. This violates the requirement of closure under learning.

If we suppose even further that it is possible to maintain closure under chunking by bounding the total number of conditions (and thus the total number of multi-attribute-matching conditions) — for example, by excluding learning over recursive constructs [23] — then relative requirements come to the fore. In particular, though the computational cost may be polynomially bounded, it may be a large enough polynomial so as to make the approach quite unattractive. The approach may also suffer significantly along the completeness dimension, as it is quite selective in the rules that it learns.

Thus, none of these *simple* unique- and multi-attribute combinations generate attractive alternatives.

5. The Space of Local Syntactic Restrictions

The unique-attribute representation restricts the triple-based representation — `(object ^attribute value)` — in just one specific way. However, there is a large range of possible restrictions on the triple representation, and any of these could potentially provide an alternative representation that meets the absolute requirements, but has better expressivity and generality than unique-attributes. In this section we investigate this space of local restrictions on the set of triples in working memory (or equivalently, on the graph structure that they form).

Consider, for instance, a restriction that disallows multiple WMEs having identical value fields, irrespective of their objects and attributes. In terms of the directed labeled graph representations of production and working memory (see Section 2.2), this restriction disallows multiple directed edges from entering (i.e., terminating in) a single node; that is, it forces all graphs to be tree-structured. Since isomorphism between tree-structured graphs can be performed in polynomial time [27], this restriction may possibly provide a polynomial bound on production match. Thus, a tree-structured restriction may potentially provide a good alternative to unique-attributes.

A second restriction that might also provide useful alternative(s) to unique-attributes, is motivated by considerations of *symmetry*. For instance, given the directed graph representation of productions and working memory, restrictions applied to the edges entering a node could also be symmetrically applied to edges leaving (or originating from) a node. When applied to the tree-structured graph mentioned above, this consideration of symmetry yields a representation that disallows multiple edges leaving any of the nodes. This may also lead to a representation with a non-combinatorial match. A similar consideration of symmetry with respect to the unique-attribute representation leads to the ***unique-object*** representation. In particular, unique-attributes eliminate WMEs which have identical object and attribute fields, but have different value fields. A symmetrical restriction would eliminate WMEs which have identical value and attribute fields, but have different object fields. Thus, this restriction would disallow having the pair of WMEs `(A ^connected C)` and `(B ^connected C)`, because they are identical in their value and attribute fields, but have two different entries in the object field: `A` and `B`. Interestingly, when the Grid task is represented with unique-attributes (as in Section 3.1), it automatically adheres to the unique-objects constraint. Thus, for the Grid task, the unique-object representation provides the same benefits as the unique-attribute representation. Thus, symmetrical restrictions may potentially also provide us alternatives to unique-attributes.

What is needed most here is a systematic way of generating these alternatives, plus any other such restrictions that may also be worth investigating. The approach we have taken is to formulate a set of independent dimensions with respect to the directed labeled graph representation of working memory. Figure 8 reproduces a portion of this graph representation from Figure 2. Consider a node in the working memory graph and the directed labeled edges leaving the node as shown in Figure 8-b. The labels on the edges can be considered as a function F of the edges. Figure 9 shows this function F. It maps an edge to its corresponding label. Since each WME has an attribute associated with it, F is defined for every edge leaving a node. Note that F is a local function, so that a new F, with its own set of edges and labels, is defined for every node in the graph. F is a function, and not a general relation, because a single edge cannot map onto multiple labels — that would imply more than one attribute in a single WME. However, F can map multiple edges onto a single label, thus denoting a multi-attribute. For instance, in Figure 9, the two edges mapped onto a single label yield the multi-attribute: `(A ^connected B) (A ^connected C)`.

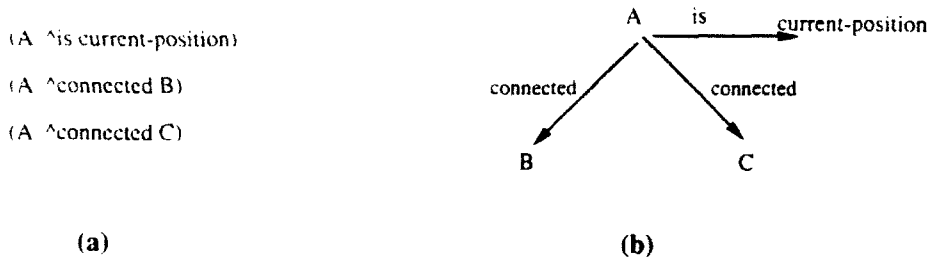


Figure 8: A few WMEs from Figure 1-b and the corresponding graph.

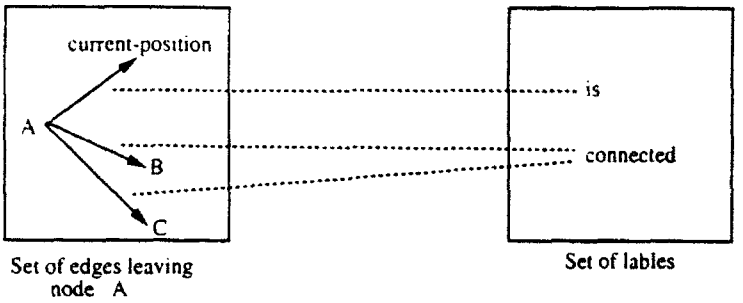


Figure 9: Labels on edges leaving a node as a function F of the set of edges.

Function F helps in systematically generating different working-memory representations. This is achieved by using two different dimensions to impose restrictions on F. A value along a specific dimension gives a specific restriction on F. This in turn yields a specific restriction on the working memory graph, and thus a specific working-memory representation. The two dimensions are motivated from basic set-theoretic considerations about F: (i) the type of the function F, i.e., whether it is one-to-one or many-to-one; and (ii) the cardinality of the range set, i.e., the cardinality of the set of labels. Other such dimensions may exist as well; however, discovering them remains an interesting issue for future

work.[7]

In more detail, the two dimensions, and the values along them, are:

1. *The type of* ***F***: We focus on two values along this dimension: one-to-one (denoted as 1-1) and many-to-one or unrestricted (denoted as *-1). If ***F*** is of type 1-1, then it forces a unique label per edge leaving a node. This is shown in Figure 10. This 1-1 restriction on ***F*** generates the unique-attribute representation, since no two edges leaving a node can have the same attribute. If ***F*** is of type *-1, then it allows multi-attributes.

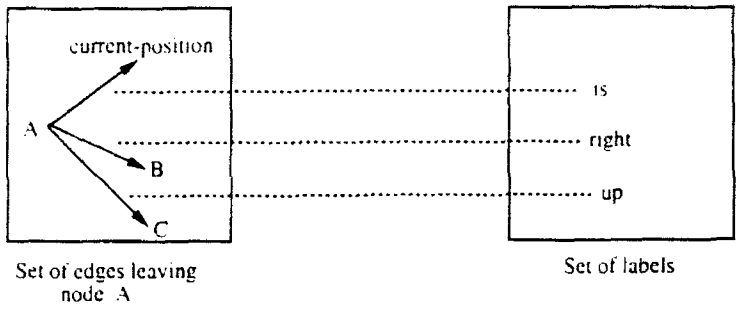


Figure 10: Restricting the function ***F*** to be one-one.

2. *Cardinality of the set of labels*: We focus on two values along this dimension: 1 and unrestricted (*). A cardinality of 1 forces the same label on every edge leaving a node, as shown in Figure 11.

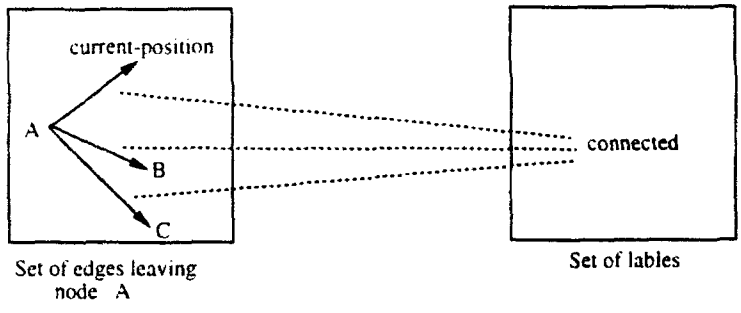


Figure 11: Restricting the cardinality of the set of labels.

Given the earlier discussion regarding symmetry of restrictions, a function ***G*** can be defined that is symmetrical to ***F***. The function ***G*** applies to edges entering a node. The corresponding two dimensions of restrictions — the type of the function and the cardinality of the set of labels — can also be defined for the function ***G***. Thus, four dimensions of restrictions on the working memory graphs are obtained, two for edges leaving a node, and two for edges entering a node. These dimensions, along with the values along them, are shown below:

1. Type of ***F*** (for edges leaving a node): 1-1 or *-1
2. Cardinality of the set of labels (for edges leaving a node): 1 or *.

---

[7]Since the cardinality of the set of labels is considered as a dimension, the cardinality of the set of edges could also serve as a dimension. However, as discussed below, the two values of interest along this dimension are 1 and unrestricted. These values can be derived from values along the other dimensions, i.e., the cardinality of the set of edges is not an independent dimension for the two values of interest.

3. Type of ***G*** (for edges entering a node): 1-1 or *-1.

4. Cardinality of the set of labels (for edges entering a node): 1 or *.

Figure 12 shows the four dimensions in a tabular format. To lay out these four dimensions in two dimensions, the dimensions for the edges leaving a node are paired. Similarly, the dimensions for edges entering a node are paired. The figure shows that different combinations of values along these dimensions identify different working memory representations. Note that to illustrate the symmetry in the combination of values, the first row and the first column are repeated.

Our exclusive focus on the values of (1, *) and (1-1, *-1) along the four dimensions, as opposed to say (2, *) or (2-1, *-1), is an interesting issue. As discussed below, the present choice of values covers all the representations discussed earlier, including unique-attributes, unique-objects, tree and the unrestricted representation. Furthermore, there is a symmetry in the set of values along the four dimensions. Thus, the choice of values along the four dimensions appears to be appropriate. However, it remains unclear whether other values might be useful.

In the unrestricted representation, there are no restrictions on the working memory graph. Thus, for edges leaving a node, the type of ***F*** is *-1, and the cardinality of the set of labels is *. Similarly, for edges entering a node, the type of ***G*** is *-1, and the cardinality of the set of labels is *. Therefore, the unrestricted working memory occupies the square in the center of the table, where the co-ordinates are [*, *-1, *, *-1]. For unique-attributes, the restriction is only on the type of ***F*** — it is restricted to 1-1. Its other dimensions are unrestricted, and thus the co-ordinates for unique-attributes are: [*, 1-1, *, *-1].

In Figure 12, the square marked as *Tree* indicates the representation discussed earlier — it restricts the working memory and production graphs to tree structures. In particular, it restricts the dimensions for edges entering a node: limiting the type of ***G*** to 1-1 and the cardinality of the set of labels (for edges entering a node) to 1, leading to the tree restriction. Its symmetrical counterpart, which restricts ***F*** to 1-1 and the cardinality of the set of labels (for edges leaving a node) to 1, is also covered in the table. The symmetrical counterpart of unique-attributes — *unique-objects* — is also covered in the table. The unique-object representation restricts the type of ***G*** to 1-1, but does not restrict any other dimension. It forces a unique label per edge entering a node, i.e., it forces the object field to be unique given identical attribute and value fields.

Thus, all the representations discussed earlier are in this table. Furthermore, some additional representations have been generated, including some that combine the restrictions from the different representations. For instance, the square [*, 1-1, *, 1-1] combines the restrictions from the unique-attributes and unique-objects representations.

This tabular organization enables drawing a variety of conclusions about these representations. First, information can be derived about how the representations relate to each other in terms of amount of restrictiveness. The table shows that the unrestricted representation is the least restrictive one, given this set of dimensions. It occupies the center square in the table. The other representations form a restriction lattice — the arrows along the side of the table show increasing restrictiveness. The most restrictive

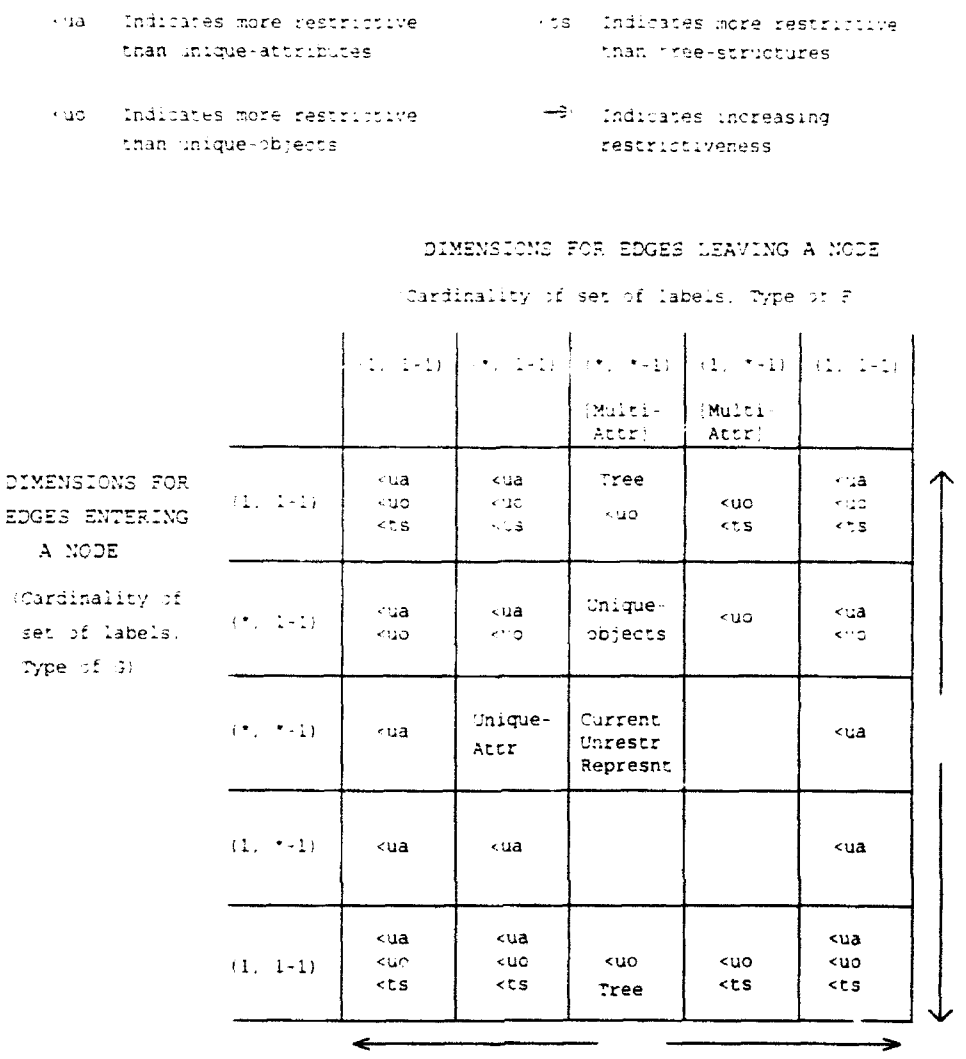


Figure 12: Dimensions of alternative representations.

working memory representation is the one with the co-ordinates [1, 1-1, 1, 1-1]. The table also shows that the Tree representation is more restrictive than unique-objects; but that unique-attributes are unrelated in terms of restrictions to either the Tree or the unique-object representation.

Second, the table shows that the Tree representation does not hold up as an alternative to unique-attributes. In particular, as modeled in the model production system formulation, this representation allows ***F*** to be *-1, i.e., it allows multi-attributes. As discussed earlier, match in the presence of multi-attributes is combinatoric — at least with instantiation match — so the tree representation violates the absolute requirement of a non-combinatorial match. This conclusion, which is clearly contrary to the earlier intuition, holds even if an optimized polynomial-time tree-matching algorithm is used. The reason is that as formulated here, the tree-representation allows the generation of a combinatorial number of instantiations from the match of a single production. The unique-object representation is also seen to allow ***F*** to be *-1, i.e., it also allows multi-attributes, and hence does not provide a non-combinatorial match bound. In fact, all the working memory representations in the third and fourth columns allow multi-attributes (they are labeled **[Multi-attr]**). These representations can now all be ruled out as

alternatives to unique-attributes.

The earlier observation about increasing restrictiveness also allows comparing unique-attributes (in the second column) with all the other representations in the first two columns of the table. This comparison can be performed using the relative requirements defined earlier. All the representation in the first two columns are more restrictive than unique-attributes. The representations that are more restrictive than unique-attributes do not reduce the match bound — since all the conditions of a production must be examined for match, the O(conditions) bound in unique-attributes is optimal to begin with. Furthermore, these representations are guaranteed not to be better than unique-attributes along the expressive adequacy or learning generality requirements, since they are more restrictive, i.e., unique-attributes subsume these representations.

Thus, the final conclusion that can be drawn from the table is that *unique-attributes provide the best possible representation within the four dimensions investigated.* That is, within this space of local syntactic modifications, unique-attributes provide the best fit to all the absolute and relative requirements. All other representations are either combinatoric, so that they violate the absolute requirement of a polynomial match bound, or they are more restrictive than unique-attributes.

6. A Space of Bounded K-Search: Instantiationless Match

There are many conceivable ways to restrict the structure of the k-search trees to guarantee a size that is polynomial in both the number of production conditions and the size of working memory. Here we focus on just one such approach, based on *instantiationless match*.

The standard instantiation match produces complete instantiations as the end result of the match. Consider the production system in Figure 1. The end result of the match is four instantiations: (<x> = A, <y> = B, <z> = D, <w> = E), (<x> = A, <y> = B, <z> = D, <w> = F), (<x> = A, <y> = C, <z> = D, <w> = E), and (<x> = A, <y> = C, <z> = D, <w> = F). Each instantiation indicates what variable bindings go together, e.g., in the first instantiation, the binding A for variable <x>, B for variable <y>, D for variable <z>, and E for variable <w> go together.

Instantiationless match is a new approach that weakens the requirements on what the match is to produce. Instead of generating separate instantiations, this match only requires that each variable obtain a set of bindings. If the production in Figure 1-a is matched with the working memory from Figure 1-b using instantiationless match, the end result will be a set of bindings for the variables as follows:

<x> = A; <y> = B, C; <z> = D; <w> = E,F.

The matcher guarantees that these bindings are *consistent* with each other. Here, consistency implies that the variables obtain the same bindings as they would have obtained in the instantiations generated via instantiation match. For instance, there are two bindings — B and C — for the variable <y> in the two instantiations generated with instantiation match for the production. Therefore, in instantiationless match, variable <y> will obtain bindings B and C.

As mentioned in Section 1, the motivation behind instantiationless match is its ability to restrict the

structure of k-search trees, and thus possibly bound match cost. Consider the example from Figure 1, the variables <x>, <y>, <z> and <w> had one or two bindings each, generating ($1 \times 2 \times 1 \times 2 = 4$) 4 instantiations. But, suppose all of these variables had N (N >>1) bindings each, then instantiation match will generate ($N \times N \times N \times N = N^4$) N^4 instantiations. However, instantiationless match will only require each variable to obtain its set of N bindings, resulting in the creation of only ($N + N + N + N = 4N$) 4N bindings overall. This may not translate into a reduction of match cost to 4N, given the requirement of consistency of bindings; but, at least, the burden of generating N^4 instantiations is eliminated. Thus, instantiationless match *could* allow the reduction of cross-products and thus potentially replace the multiplicative process in instantiation match by an additive process. In terms of the structure of the k-search tree, instantiationless match alleviates the cascading (or cross-product) effect in the k-search tree — multiple WMEs may match a condition, but their branching effect is avoided.

While this shift in the nature of match from instantiation match to instantiationless match appears promising, it still needs to be evaluated with respect to the overall objective. Specifically, (i) whether a production system can function without instantiations, (ii) whether this shift guarantees a non-combinatorial match bound, and (iii) what tradeoffs it implies. The first two of these issues are addressed in the remainder of this section. The third issue is then covered in the next section, but only with respect to a particular combination of instantiationless match and a tree-structured restriction on production conditions.

6.1. Functioning with Instantiationless Match

Instantiations are unnecessary in a production system as long as there is no change in either the (RHS) actions produced or the times at which they are produced. This is what instantiationless match aims to achieve. It simply changes the computation of the bindings for the variables in the actions. As discussed in Section 2.2, with instantiation match, bindings for actions are obtained from individual instantiations. With instantiationless match, bindings are obtained directly from the bindings for individual variables. In Figure 1-a, the actions produced with instantiationless match would be the same as the actions produced with instantiation match — the variables <x> and <w> obtain identical bindings in both cases.

However, in general, producing identical actions with instantiationless match may appear unworkable. This is mainly due to the problem of *binding confusion* on the action side. Figure 13 illustrates this problem. Figure 13-a shows a production that adds the relation `^dist-2` between the bindings of <c1> and <c3>. Given the working memory in 13-b, instantiationless match results in two bindings each for the variables <c1> and <c3>, as shown in 13-c. Therefore, this production firing will result in adding a `^dist-2` attribute between the following pairs of symbols: (C1, C3), (C1, C6), (C4, C3), and (C4, C6). In contrast, with instantiation match, two instantiations — (<s> = S1, <c1> = C1, <c2> = C2, <c3> = C3), and (<s> = S1, <c1> = C4, <c2> = C5, <c3> = C6) — would be generated. When they fire, the attribute `^dist-2` would be added only between (C1, C3), and (C4, C6).

A closer inspection reveals that binding confusion is not quite as problematical as it may at first seem. First, only a few of the variables from the condition side actually appear on the action side, i.e., the action

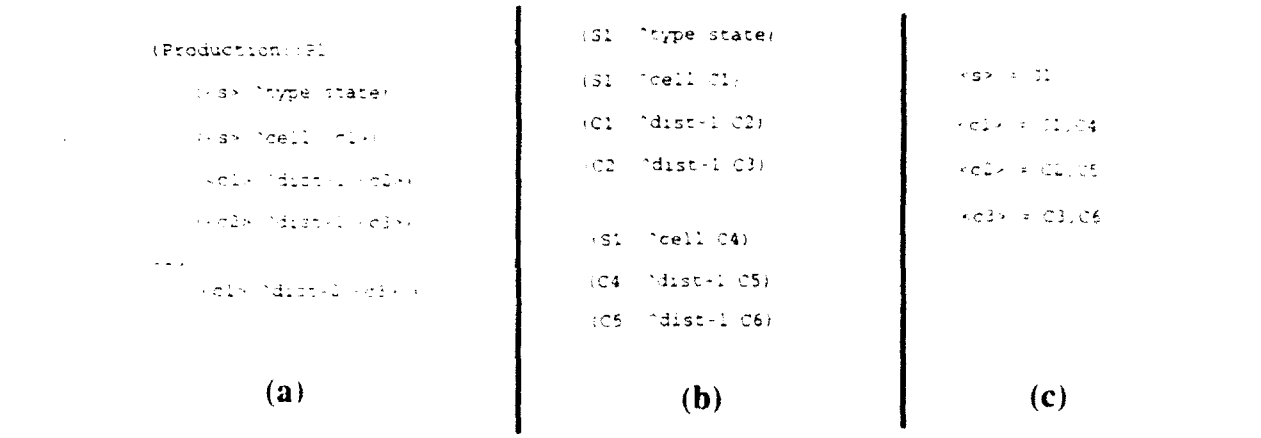


Figure 13: The problems of binding confusion in instantiationless match: (a) a production, (b) WMEs that match the production, (c) the result of instantiationless match.

side requires only a small subset of variable bindings. Second, binding confusion can arise only in the presence of multiple bindings for multiple of the variables involved. For instance, in Figure 13, both variables <c1> and <c3> have multiple bindings. If only one of the variables had multiple bindings, then only two actions would be possible, and the result of instantiation and instantiationless match would be identical. Indeed, in a small set of tasks described later, variables on the action side are seen to have single bindings, and thus binding confusion is not an issue.

While this approach has proven adequate for the tasks investigated here, it clearly isn't sufficient for guaranteeing correct performance across arbitrary tasks. One possibility for providing such a guarantee is a hybrid system that keeps instantiation information just with respect to those variables that occur in both the conditions and actions. If the number of variables that could be so shared was then bounded (e.g., to 2), then it may be possible to tractably eliminate binding confusion without seriously impacting expressibility. A second possibility is to order the bindings for the variables in some pre-determined way [54]. For instance, in Figure 13-c, a pre-determined ordering on the bindings for variables could be exploited to put C1 (the first binding for <c1>) in correspondence with C3 (the first binding for <c3>), and analogously C2 in correspondence with C4. This would allow production firings to add the attribute ^dist-2 only between (C1,C3), and (C2,C4). Essentially, this ordering maintains some of the instantiation information. A third possibility is to extend the semantics of the production system so that it can handle sets in the object and value fields of WMEs. If this could be done, and yield a sufficiently functional system, then binding confusion could simply be ignored, as the use of a variable in an action could simply mean that that variable should be instantiated with the the set of possible bindings, rather than with individuals values from the set. A fourth possibility that has been suggested is to replace the use of equality tests in the conditions of productions with an equality testing function on the action side. However, this turns out to not help at all — if there are multiple bindings for variables, then, given the absence of instantiations, the system cannot pinpoint which pairs of bindings should be associated with each other, and therefore which pairs should be tested for equality. A more detailed investigation of this space of possible general solutions to the binding confusion problem is left for future work.

A second apparent problem with instantiationless match arises from the *conflict-resolution* phase embedded in some production systems, e.g., OPS5 [15]. In OPS5, conflict resolution immediately follows match. It arbitrates among different production instantiations to select one, which is subsequently fired. A variety of strategies for conflict-resolution are possible, but at least some of these cannot function without instantiations. However, the key point to note here is that conflict-resolution strategies are often simple heuristics that substitute for more principled reasoning [80], and they have been criticized on a variety of counts — from causing maintainability problems [80] to being inadequate as the basis for integrated AI architectures [59]. In fact, systems such as Soar (and Prodigy [50]) have already abandoned OPS5-style conflict resolution in favor of more principled reasoning, and thus do not require specific production instantiations. However, for systems that do depend on instantiations, the heuristics in the conflict-resolution phase would need to be modified so as to suit instantiationless match.

Thus, instantiationless match is not as problematical as it may appear at first. Some difficulties remain to be resolved — binding confusion in particular — but they do not appear to be insurmountable.

6.2. The Complexity of Instantiationless Match

On the second issue, unfortunately, instantiationless match by itself is unable to guarantee a non-combinatorial match bound. One way to understand why this is true is to map instantiationless match onto constraint-satisfaction problems. This mapping allows tapping into a rich vein of existing results. Some of these results bear directly on this issue of the combinatoric nature of instantiationless match. However, other results go beyond this to reveal additional restrictions that may yield polynomial match algorithms when combined with instantiationless match, as well as algorithms for performing these matches[8].

A constraint-satisfaction problem is defined as follows: given a set of N variables each with an associated domain and a set of binary constraining relations between the variables[9], find all possible N-tuples such that each N-tuple is an instantiation of the N variables satisfying the constraining relations [45]. This problem can be represented as a *constraint graph* where the variables are represented by nodes and the constraints by arcs. Each constraint specifies the set of permitted pairs of values for the two variables involved. Thus, if X_i and X_j are two variables with domains D_i and D_j respectively, then the constraint R_{ij} between X_i and X_j is a subset of the cartesian product of their domains, i.e., $R_{ij} \subseteq D_i \times D_j$

The match for a single production maps on to the constraint-satisfaction problem as follows. The variables in the conditions form the variables in the constraint-satisfaction problem. For example, the production in Figure 1-a can be represented as the constraint graph in Figure 14 (this is same as the production graph in Figure 2-a). The symbols occupying the identifier and value fields of WMEs form the domains of the variables. A condition containing two variables forms an arc between the two

---

[8]Others have also started to investigate constraint-satisfaction algorithms in the context of production match [63], but not specifically to bound the cost of match.

[9]In general, the constraints need not be limited to being binary; however, in this article, only binary constraints are considered.

variables in the constraint graph. The condition specifies (or selects) the WMEs with its attribute, so that each WME represents a permitted pair of values for the variables linked by the condition. Thus, the condition, and the WMEs it selects, form a constraint between the two variables linked by the condition. Figure 15 shows the conditions in the production with the permitted pairs of values for the variables linked by the conditions, given the working memory in Figure 1-b.[10] Resolving the conflict between the possible values for variables and finding all possible solutions of the constraint satisfaction problem formed by a production will result in finding all possible instantiations of the production using instantiation match.

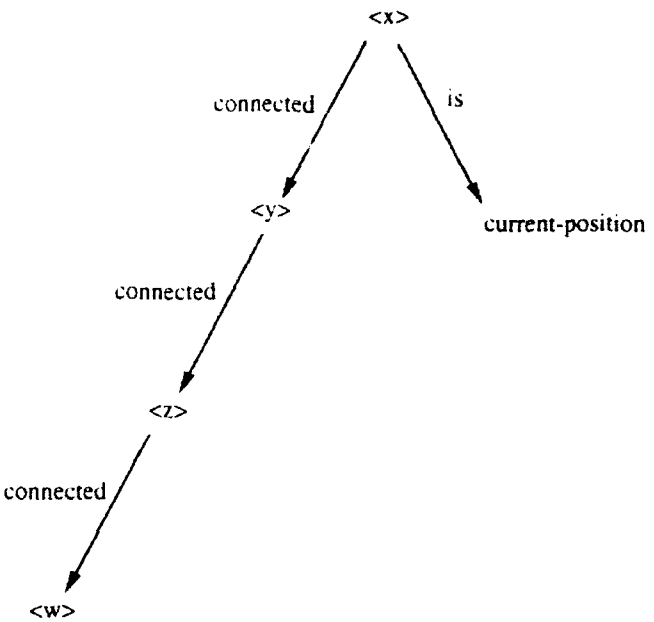


Figure 14: Viewing a production as a constraint graph.

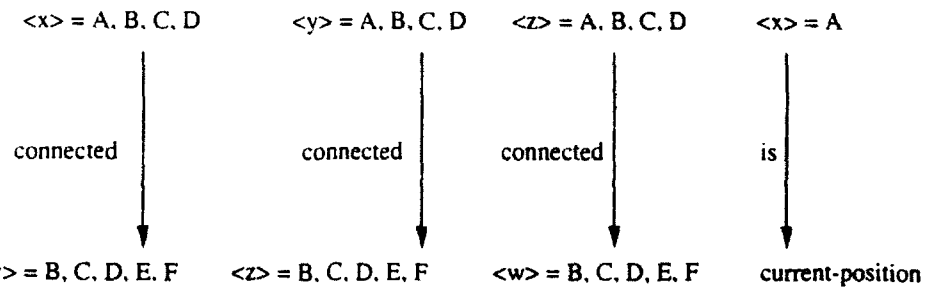


Figure 15: Permitted pairs of values for conditions, given the WMEs from Figure 1-b

In contrast with production-match algorithms, which check the consistency between bindings while forming tokens and instantiations, many constraint satisfaction algorithms distinguish the two phases of obtaining consistent bindings and forming instantiations [20]. Once consistent bindings are obtained, individual instantiations from the bindings are obtained separately. Instantiationless match computes an

[10]For a negated condition of the form `-(<x> ^attr <y>)`, the domains of <x> and <y> consist of the symbols in the identifier and value fields of all the WMEs except those of the form `(A ^attr B)`.

identical consistent set of bindings, without forming any instantiations. In the example from Figure 14, instantiationless match achieves the bindings <x> = A; <y> = B,C; <z> = D; <w> = E,F.

Thus, there is a mapping between instantiationless match and obtaining consistent bindings in constraint satisfaction problems. This mapping directly provides a number of results in terms of implementations, algorithms, and complexity analyses. Given a focus on polynomially bounded match, the complexity results in the constraint satisfaction literature are of particular interest. The basic result here is that obtaining consistent bindings for the variables in an unrestricted constraint graph is NP-hard [20]. This result implies that if the production graph is not restricted, then instantiationless match remains NP-hard.

Results from the constraint satisfaction literature also show that by restricting a constraint graph in specific ways, it is possible to obtain consistent bindings for its variables in polynomial time. In terms of instantiationless match, these results indicate that by restricting the production graph in different ways, different non-combinatorial bounds on instantiationless match can be obtained. For instance, the graph in Figure 14 is a tree. It is possible to obtain consistent bindings for its variables in polynomial time using the *arc-consistency* algorithm [20]. A single instantiation can then be generated from these bindings in time linear in the number of variables [26]. Generating this single instantiation requires collecting the bindings for variables in a particular pre-determined order. However, generating all instantiations could be a combinatoric process, since the variable bindings could result in a combinatorial number of instantiations.

Thus, by restricting the match to be instantiationless, and by restricting the production graph in different ways, it is possible to obtain different polynomial bounds on production match.

7. Instantiationless Match with Tree-structured Productions

In the previous section it was shown that a combination of instantiationless match and tree-structured production graphs — even with no restrictions on the structure of working memory — can yield a polynomial bound on match when the arc-consistency algorithm is exploited. In this section, this result is developed in more detail as the basis for a serious alternative to unique-attributes, called the *instantiationless-tree* (or *I-Tree*, for short) approach[11].

In constraint-satisfaction problems, arc-consistency is a form of local consistency. It does not solve the general constraint-satisfaction problem. A constraint graph is arc-consistent if each of its arcs is arc-consistent. Suppose X_i and X_j are two nodes of the constraint graph with domains D_i and D_j respectively, and an arc R_{ij} between X_i and X_j. This arc is arc-consistent iff for any value $x \in D_i$ there is a value $y \in D_j$ such that $R_{ij}(x, y)$ [44]. Here $R_{ij}(x, y)$ stands for the assertion that (x, y) is permitted by the explicit constraint R_{ij}. If R_{ij} is a directed arc then arc-consistency requires the following in addition: if there is a value $y \in D_j$ then there is a value $x \in D_i$ such that $R_{ij}(x, y)$ [20].

---

[11] In [72], the computation in the I-tree approach was shown to be equivalent to the computation in the NETL parallel marker-passing system [24].

In terms of the mapping, the following has to be satisfied for a condition (<x> ^A1 <y>) to be arc-consistent: If there is a binding X for the variable <x> in its identifier field, then there is a binding Y for the variable <y> in its value field, such that there exists a WME with attribute A1, identifier X and value Y, i.e., (X ^A1 Y). Since conditions in a production graph form directed arcs, the following is required in addition: if there is a binding Y for the variable <y> in its value field, then there exists a binding X for the variable <x> in its identifier field such that there exists a WME (X ^A1 Y). A production is arc-consistent if each of its conditions is arc-consistent.

An important result stated in [20] is that arc-consistency achieves consistent bindings for tree-structured constraint graphs, even in the absence of any restrictions on domains of variables (Theorem 2.4). Given the mapping of constraint graphs on to production structure, the following corollary can be derived:

> **The I-Tree Corollary:** If a production is tree structured, then arc-consistency provides consistent bindings for the variables in the production. That is, for tree-structured productions, instantiationless match can be achieved using arc-consistency. No restrictions on the working memory representation are required. Furthermore, this arc-consistency can be performed in O(*#WMEs*#conditions*).

Here, *#WMEs* refers to the number of working memory elements and *#conditions* to the number of conditions in the LHS of a single production.

An algorithm has been derived for instantiationless match over tree-structured productions [70] that is directly based on the algorithm for arc-consistency presented in [20]. The complexity analysis of this algorithm directly follows from its analysis in [20].

To understand this all better, let's briefly examine why each of the two restrictions — arc-consistency match and tree structured productions — are individually not sufficient for achieving a non-combinatorial bound on match. First, consider tree-structured productions with instantiation match instead of arc-consistency match. Instantiation match, despite tree-structured productions, is still exponential. This is illustrated by the tree-structured production in Figure 6-b, which leads to an exponential match. Second, consider arc-consistency match with non-tree-structured productions. Here, arc consistency may provide a wrong result. That is, it may provide a binding for a variable, when instantiation match for the same production would have provided none.

Figure 16 illustrates this last point with the help of an example. Figure 16-a shows a production in the system. This production is non-tree structured, as shown by the constraint graph of the production appearing in Figure 16-d. The conditions of the production test if an operator is stacking a block on top of itself. The action of the production rejects that operator from consideration. Figure 16-b shows the working memory in the system. There are two operators in working memory, none stacking a block on itself. Figure 16-c shows the outcome of the instantiation match. The match for the fourth condition fails, and there are no instantiations of the production, and no operator is rejected.

However, the arc-consistency match for the production succeeds, resulting in the bindings for the variables as shown in 16-d. To understand how these bindings make the production arc-consistent, consider the condition (<o> ^stacks-block <b1>) in the production. The WMEs (O1 ^stacks-block B1) and (O2 ^stacks-block B2) provide the bindings O1 and O2 for the variable <o> and B1 and B2 for the variable <b1> according to the definition of arc-consistency.

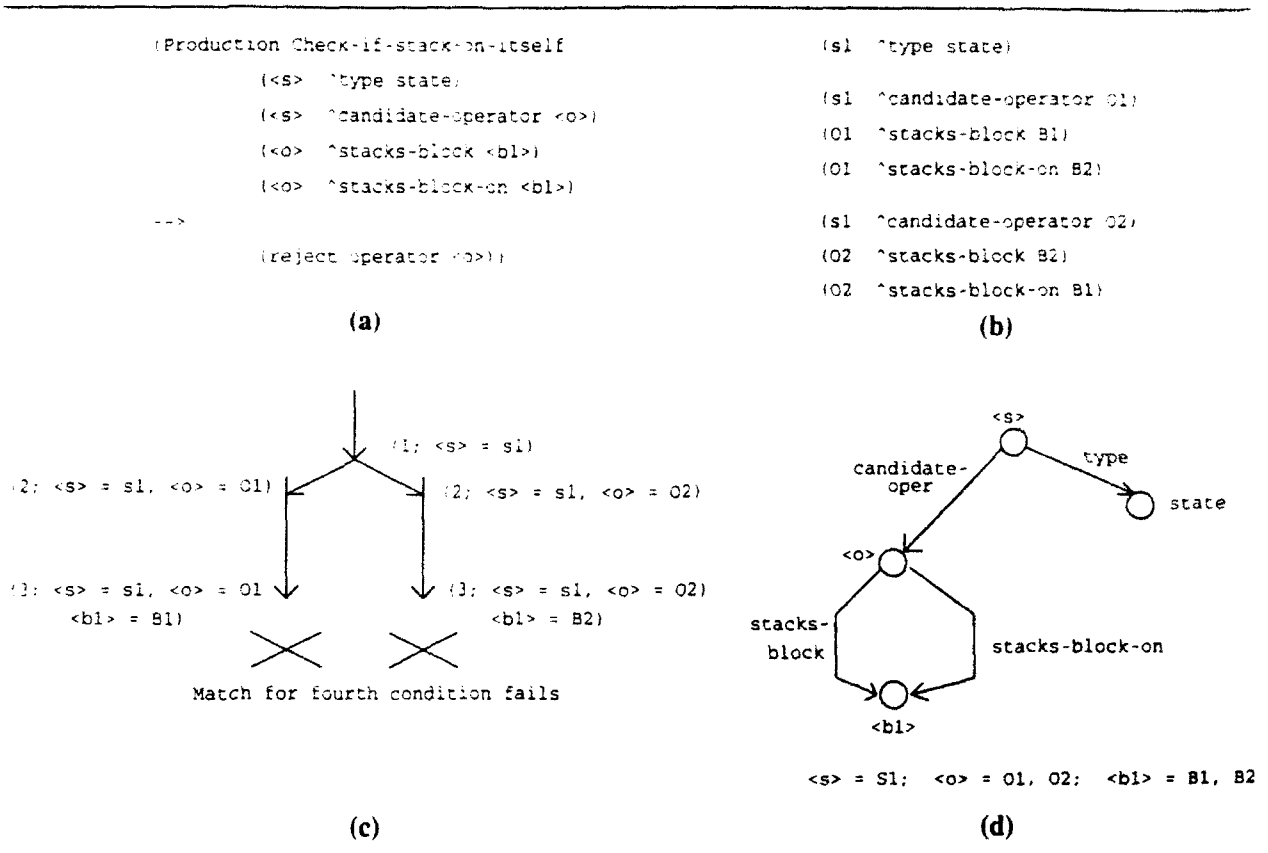


Figure 16: Arc-consistency match provides incorrect results for non-tree structured productions: (a) a production, (b) WMEs to match the production, (c) the result of instantiation match, (d) the result of arc-consistency match.

Similarly, consider the condition `(<o> ^stacks-block-on <b1>)`. The WMEs `(O1 ^stacks-block-on B2)` and `(O2 ^stacks-block-on B1)` provide the bindings O1 and O2 for the variable `<o>` and B1 and B2 for the variable `<b1>`. The key point to note here is that for both conditions, the two variables obtain the bindings without maintaining information about which binding is consistent with which other binding. Thus, for the first condition, O1 is consistent only with B1, while for the second condition, O1 is consistent only with B2. This lack of information allows the arc-consistency match to succeed (it is easy to verify that the other bindings make the remaining conditions in the production arc-consistent). Since this arc-consistency match succeeds in providing bindings for all of the variables, operators O1 and O2 are both rejected. Thus, arc-consistency match may fail to provide consistent bindings for non-tree-structured productions (as per the definition of consistency from Section 6), i.e., the tree-structured restriction is necessary in achieving consistent bindings.

It is easy to confuse this last issue with the problem of binding confusion; however, they are in fact quite distinct. Binding confusion assumes that you already have a consistent solution to the constraint satisfaction problem defined by the production conditions. The problem there is that, given these consistent bindings, it is difficult to determine which of the individual bindings of one variable are consistent with which of the individual bindings of the other variables. This information is not needed for

the match, but is needed for action. On the other hand, the problem described here is whether the arc-consistency-based match algorithm even returns consistent sets of bindings in the first place.

8. Evaluating the Instantiationless-Tree Formulation

The appropriate way to evaluate I-Tree is according to the absolute and relative requirements presented in Section 4. The first absolute requirement is that of a polynomial bound on production match. Section 7 indicates that I-Tree achieves a polynomial bound of O(*#WMEs*#conditions*). I-Tree also meets the requirement of correctness of match in that it provides consistent bindings for variables (assuming tree-structured productions). It also meets the requirement of closure under learning. Chunking, as currently defined, does not provide such closure; however, closure is provided by closely related EBL algorithms that avoid introducing equality tests where they did not already exist in the original rules. Finally, it also meets the requirement of expressibility as a local syntactic restriction: if a single variable appears in the value field of two conditions, there is a violation.

With respect to the relative requirements, I-Tree must be compared with something. Here the focus will be on comparing it with the only other active proposal for eliminating combinatorics, i.e., unique-attributes; however, when appropriate, a few words will also be said with respect to the unrestricted representation. The comparison will be performed in three steps. First, the Grid Task will be examined in some detail, primarily to illustrate the gains in generality that I-Tree makes possible over unique-attributes. Second, the expressive adequacy of I-Tree will be investigated in some depth. Third, the results from these first two steps will be combined with results on the other relative requirements to generate a summary comparison between the two approaches. The first two steps will be described in this section. The next section will then focus on the third step.

8.1. The Grid Task in I-Tree

The Grid task, along with its encodings in the unrestricted and unique-attribute representations, was introduced earlier in Section 3.1. For this task, the unrestricted encoding turns out to work without change for I-Tree. The arc-consistency algorithm works because all of the productions implementing this task already meet the tree-structured restriction, and no binding confusion results because key action variables always have at most one binding. The rule learned from I-Tree also turns out to be identical to the rule learned in the unrestricted representation (Figure 6-b).

Table II expands Table I by showing the cost and generality of the learned rule in I-Tree. While the cost is measured in number of tokens for the unrestricted and unique-attribute systems, the cost for I-Tree is measured in terms of *binding operations*, i.e., the operation of obtaining a consistent binding for a variable (or deleting an inconsistent binding). The computation involved in this binding operation is comparable to the computation involved in generating a token [70].

The cost of match in I-Tree is seen to be a small polynomial, comparable to the cost of unique-attributes. As with unique-attributes, I-Tree is able to obtain this performance improvement over the unrestricted representation by avoiding the computation of unnecessary information. In particular, by not computing all of the instantiations, I-Tree avoids deriving all possible paths from the source to the

destination (while the unrestricted representation unnecessarily derives all of these paths). However, unlike unique-attributes, I-Tree is able to reduce its match cost without a change in the working-memory representation, and without the concomitant losses in the generality of the learned rules. Thus, I-Tree outperforms the unrestricted representation in performing the Grid task, while simultaneously achieving a better learning generality than unique-attributes. (A similar analysis of the tree task, with similar conclusions, appears in [70]. The tree task is identical to the grid task, except that the underlying graph is a tree rather than a grid.)

	Match cost per rule	Generality in number of transfers	Number of rules learned for same level of generality	Match cost with all rules
Unrestricted	$(4^{p+1}-4)/3$	$n^2*(p+1)^2$	1	$(4^{p+1}-4)/3$
Unique-attributes	p	n^2	$(p+1)^2$	$(p+1)^2*p$
I-Tree	$6p^3$	$n^2*(p+1)^2$	1	$6p^3$

Table II: The cost and generality of various representations for the Grid task with a fixed path length p. Cost for the unrestricted and unique-attribute systems is measured in terms of tokens. In I-Tree cost is measured in terms of binding operations.

An arc-consistency matcher has been implemented to verify the results of this analysis. Figure 17 compares the performance of this arc-consistency matcher with a Rete matcher using the unrestricted representation for this production from the grid task. A 5x5 grid is assumed, and match cost for different path lengths is measured. The figure confirms that with increasing path length, the cost of instantiation match increases exponentially, while the cost of arc-consistency match increases at a much slower rate.

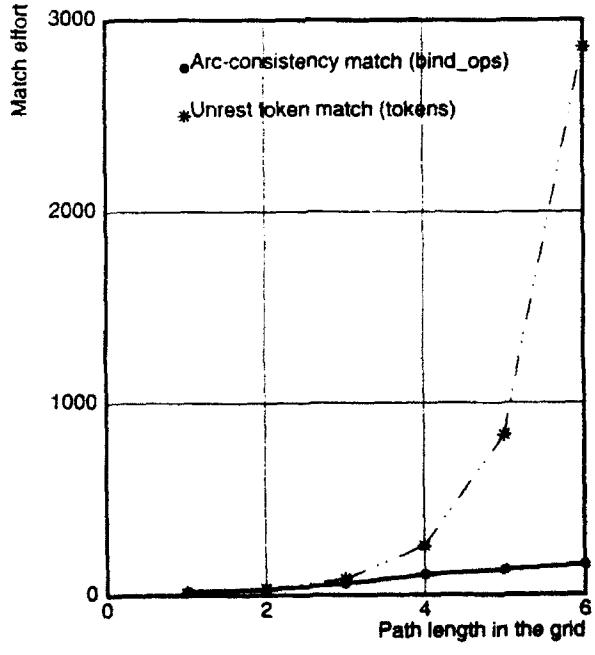


Figure 17: Comparing the cost of arc-consistency match with instantiation match in the grid task (in binding operations for arc-consistency match and tokens for instantiation match).

This arc-consistency matcher is still quite preliminary. In particular, such optimizations as state-saving and sharing [25] will need to be added before this matcher can compete effectively time-wise with

optimized instantiation-match implementations. Due to the lack of such an optimized algorithm, and the complexity of integrating such an algorithm within Soar (the tightly integrated nature of Soar makes it difficult to arbitrarily replace components [74]), we do not currently have an implementation of Soar with arc-consistency match.

8.2. Expressive Adequacy

Two aspects of expressibility are of potential concern in I-Tree. The first aspect is the inability to use variables to compare two value fields for equality, as mandated by the tree-structured production restriction. The second aspect is the need to restrict variable bindings so that binding confusion does not occur.

To understand the extent to which these two problems are likely to occur, and to enable further direct comparisons with unique-attributes, five tasks that had earlier been implemented in unique-attributes — Eight puzzle, Grid, Farmer, Waterjug and Blocks-world — have now been converted over to I-Tree.[12] With respect to equality tests across value fields of conditions, one of the tasks (Grid) has none, and hence there is no need to re-encode it in I-Tree. However, in the remaining four tasks, 50-80% of the productions do have equality tests across value fields of conditions, and thus do require re-encoding. With respect to binding confusion, there turns out to be no issue in any of the five tasks. Essentially, the *copy-and-constrain* method [69] (described below) used to generate tree-structured productions increases the likelihood of single bindings for key variables in productions.

So, the problem that really needs to be addressed with respect to these tasks is that of eliminating variable tests across value fields of conditions. An example production (P1) embodying such an equality test can be found in Figure 18. In I-Tree, such productions must be replaced. One simple method is to perform a form of *copy-and-constrain* [69], in which troublesome variables are replaced with tests of constants from the variables' domains. For example, in Figure 18, the production P1 can be replaced by the two productions P2 and P3, assuming the domain of variable <r> consists of only the two values red and green. Note that, with a good match algorithm, a larger number of productions does not necessarily imply a slowdown; for example, even with a larger number of productions, unique-attribute systems are more efficient than unrestricted systems.

```
(Production P1
    (<o>  ^color <r>)
    (<o>  ^code <r>)
 -->
(write "<o> is coded"))
```

```
(Production P2
    (<o>  ^color RED)
    (<o>  ^code RED)
 -->
(write "<o> is coded"))
```

```
(Production P3
    (<o>  ^color GREEN)
    (<o>  ^code GREEN)
 -->
(write "<o> is coded"))
```

Figure 18: Replacing variable tests to form tree-structured productions.

For two of the four problematic tasks — Waterjug and Farmer — re-encodings them via copy-and-constrain proved successful. For the other two — Blocks-world and Eight puzzle — the conversion could

[12]Given the absence of a Soar implementation with instantiationless match, we have resorted to simulating the restrictions in Soar's unrestricted system. While this does eliminate the possibility of directly measuring time or binding operations, other useful analyses are still possible.

be done, but it resulted in big losses in generality of learning. For instance, the Eight puzzle showed an approximately 60-fold increase in the number of learned rules (as opposed to the 6-fold increase with unique-attributes). For these two tasks, more complex schemes are required to improve the generality of the learned rules. For instance, in the Blocks-world, referring to blocks using indexical-functional attributes [2], such as *block-on-top-of-tower*, and *block-at-the-bottom-of-tower*, and using these to replace equality tests leads to a better generality of learning than referring to blocks as *B1*, *B2* etc.

There are at least two critical issues with respect copy-and-constrain that don't come up in these tasks, but may be critical in other tasks. First, copy-and-constrain leads to writing as many productions as there are values in the domains of variables tested for equality (variables that are not tested for equality across value fields need not be replaced). This could be quite problematical for variables with very large domains. For instance, replacing the equality test in a production that checked whether the length of two objects is the same — where the length of the objects ranges over the natural numbers — is impossible. Second, it creates a combinatorial number of productions in situations where there are multiple equality tests per production. For instance, if P1 has N equality tests, then it will require 2^N replacement productions.

The first problem can be solved by working with the *primitive* elements in a domain. Equality tests over complex elements can be performed by testing equality of these primitive elements. In the domain of natural numbers, these primitive elements are the digits 0,1...9. Only 10 productions are required to test equality of these individual digits. Consider the example of testing if two objects have the same length. This primitive-based scheme would lead to a digit-by-digit comparison of the numbers encoding the lengths of the two objects. If the two numbers have N digits each, then N production firings would be required to complete this comparison. A comparison of primitive elements may appear a high price to pay for replacing equality tests. However, such a comparison is also a key requirement in some schemes for knowledge-level learning with EBL [64], so if it is required there, nothing additional must be added to support this use. Furthermore, in a learning system, such as Soar, these comparisons can be automatically compiled into new rules that directly test for the N different digits in the two numbers involved. If two objects with these same lengths are ever tested again for equality, the learned rules can fire, eliminating the need for the sequential comparison. However, these learned rules are specific to the specific objects lengths tested for equality. Thus, a very large number of rules will be learned while performing these equality tests.

A variation on this same primitive-based strategy may also be used to address the second problem above of combinatorial number of productions. The single production with N tests can be converted to a sequence of N productions with a single test each. These productions would then fire in sequence and perform the tests in N recognize-act cycles. Each of these N productions would then require only two replacements, so a total of $2*N$ replacement productions would be needed.

In conclusion, all five of the tasks could be encoded in I-Tree; however, the expressibility limitations did severely impact two tasks, and less severely impact two additional tasks. Therefore, improving expressibility beyond what is provided here would still be desirable.

8.3. Summary Comparison of I-Tree and Unique-attributes

Ideally, when comparing two alternatives, one would dominate the other. When this happens, the dominated alternative can simply be eliminated from further consideration. When no dominance relationship can be established, there is extra impetus to search further for a new alternative that has (at least) the best attributes of both of the existing alternatives. Unfortunately, with respect to a comparison between I-Tree and unique-attributes, we are in the latter situation. Table III summarizes where these two alternatives stand with respect to the relative requirements.

Relative requirement	Unique-attribute	Instantiationless-tree
Expressive adequacy	Difficulty in encoding due to elimination of unstructured sets from working memory	Difficulty in encoding due to elimination of instantiations and equality tests from productions
Relative efficiency	Match bound O(conditions)	Match bound O(WMEs*conditions)
Learning generality	Loss due to lack of unstructured sets	Loss due to lack of equality tests
Completeness	No violations	No violations
Uniformity	No violations	Productions are tree-structured but not working memory

Table III: Relative comparisons between instantiationless-tree and unique-attributes.

The table shows that: (1) the two approaches are equal with respect to completeness; (2) unique-attributes dominates I-Tree with respect to relative efficiency and uniformity; and (3) expressive adequacy and learning generality yield no obvious dominance relationship. To get a better feel for these last two requirements, it is useful to go down a level of detail, and compare unique-attributes and I-Tree on a task-by-task basis. This comparison, as shown in Table IV, provides concrete examples of tasks that are performed better in I-Tree than in unique-attributes, and vice versa.

Tasks better in unique-attributes	Tasks better in I-Tree
Waterjug	Grid
Farmer	Eight-puzzle
Blocks-world	

Table IV: Task partitioning according to whether they are performed better in unique-attributes or instantiationless-tree.

Three of the tasks — Waterjug, Farmer and Blocks-world — are performed better in unique-attributes than in I-Tree. For all three of these tasks, the I-Tree representation ends up being very similar to the unique-attribute representation. For example, the encoding strategy adopted for the Blocks-World in I-Tree — based on indexical-functional attributes such as `^block-on-top-of-tower` — is exactly like the strategy used in unique-attributes. In particular, with unique-attributes, the blocks are represented as `(S1 ^block-on-top-of-tower B1)` etc, where S1 is the state. Thus, the three tasks are encoded in a similar fashion in the two representations schemes. However, due to the absence of explicit equality tests across condition values in I-Tree, the tasks are harder to encode within this scheme (and

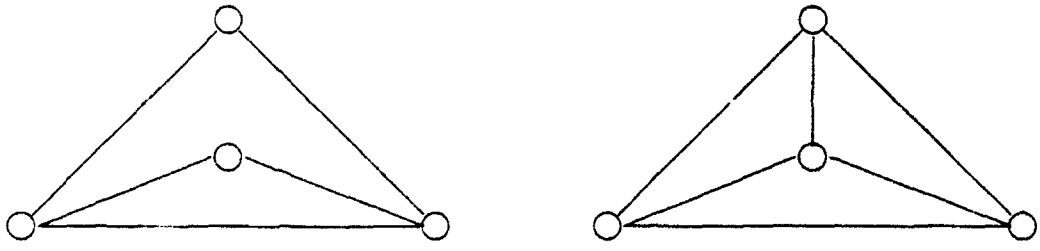
yield less learning generality).

The other two tasks — Grid and Eight puzzle — are performed better in I-Tree than in unique-attributes. As already discussed, the Grid task can proceed in I-Tree with the unrestricted encoding, which makes it easy to encode, and yields a high level of learning generality. With the Eight puzzle, while both unique-attributes and I-tree require re-encoding, I-tree obtains a better learning generality. This situation is explained in detail in [70].

Thus neither representation dominates, and the search must continue for better alternatives. This may involve hybrid approaches between unique-attributes and I-Tree, or it could involve other novel approaches, either within the spaces already delineated, or in other spaces.

9. Towards a Step Beyond Instantiationless-Tree

One particularly simple step beyond I-Tree can be found by delving further into the constraint-satisfaction literature. In particular, if instead of restricting productions to tree structures, the less restrictive class of *partial-2-trees* [20] is used, some equality tests can be allowed across value fields of conditions, while still guaranteeing a polynomial match bound. Figure 19-a shows an example partial-2-tree, while Figure 19-b shows a contrasting example of a non-partial-2-tree. Partial-2-trees are constructed by first creating a *2-tree*. A 2-tree starts out with two nodes connected by an edge. Then any time a new node is added, it is connected to two previous nodes that are already connected by an edge. Thus, the graph gets built up in triangles. Any subgraph of such a 2-tree is a partial-2-tree. These graphs are strictly more expressive than trees. In particular, because they do allow some convergence of paths on nodes, they do allow some testing of equality of value fields. An algorithm for recognizing partial-2-trees appears in [20].



(a) A partial-2-tree graph

(b) A non-partial-2-tree graph

Figure 19: Illustrating partial-2-trees.

Polynomial match with production graphs structured as partial-2-trees — and, as with I-Tree, no restrictions on working memory — can be achieved using the slightly more expensive (although still polynomial) *path-consistency* algorithm for determining consistent sets of variables bindings.

The expressive power of partial-2-trees is illustrated in Table V. The table indicates the number of

productions (both hand-coded and learned) in the unrestricted encodings of Eight puzzle, Grid, Farmer, Waterjug and Blocks-world that were found to be partial-2-trees. In I-Tree, 50-80% of the hand-coded productions and 100% of the learned productions in the Eight-puzzle, Waterjug, Farmer and Blocks-world were seen to be non-tree-structured. Only the Grid task contained all tree-structured productions. Compared with that, only a small percentage of the productions in these five tasks violate the partial-2-tree restriction. Thus, in terms of expressiveness, partial-2-tree productions are quite powerful, at least in comparison with tree-structured productions. The only significant expressibility limitation is found in the learned rules for the Blocks-world, where 80% are non-partial-2-tree.

Task	Total Number of productions tested	Percentage of hand-coded rules that are partial-2-tree	Percentage of learned rules that are partial-2-tree
Grid	22	100	100
Eight-puzzle	28	88	84
Waterjug	29	89	100
Farmer	34	100	100
Blocks	84	84	20

Table V: Percentage of hand-coded and learned productions found to be partial-2-tree in five tasks[13]

Though partial-2-trees thus look like an interesting possibility, much remains to be understood about them. Can the productions that don't meet the restriction be converted without sacrificing too much generality? Can the approach be combined with an effective solution to the binding-confusion problem? Is there a simple, local, syntactic guideline that programmers can follow in writing productions that meet the restriction (trees support this, but the published algorithm for determining partial-2-treeness does not)? Will the match algorithms be efficient in practice? Answering such questions is critical future work.

10. Summary

Eliminating combinatorics from production match is important for a number of areas of AI, including expert systems, real-time performance, machine learning, parallel implementation, and cognitive modeling. In [71], the unique-attribute representation was introduced to eliminate these match combinatorics. However, unique-attributes engender sufficiently problematic representational tradeoffs that there is a strong motivation to investigate alternative representations.

This article has focused on exploring such alternative representations. Two different spaces of alternatives have been characterized and explored. The first space consists of local syntactic restrictions on working memory. Within this space, it was shown that unique-attributes are the best possible restriction. The second space consists of restrictions on match-search (i.e., k-search) trees that guarantee polynomial boundedness. Focus here was limited to a subset of this space based on a new approach to

[13]Except for the Blocks-world, all of the productions in all of the tasks were tested. In Blocks-world, all of the hand-coded productions were tested, but since there was a large number of learned productions, only 10 representative ones were tested.

match, called instantiationless match. Through a mapping onto constraint-satisfaction algorithms, several representational restrictions have been derived that, when combined with instantiationless match, guarantee polynomially bounded match. Such alternatives show distinct promise by outperforming unique-attributes in some domains; however, no such alternative has yet been found that completely dominates unique-attributes in all domains. A few problems, such as binding confusion, also remain outstanding with these alternatives. Thus, though significant progress has been made in generating and evaluating the set of possible alternatives, the investigation of non-combinatorial match formulations must still continue.

If we look beyond our own goal of eliminating combinatoric production match, the results presented here may also have significant implications for the weaker goal of speeding up standard (combinatoric) production systems. In recent years, considerable research effort has been focused on this weaker goal, much of it through the development of match algorithms such as Rete [25], modified versions of Rete [30, 8], Treat [53], LEAPS [55], Matchbox [61], Scaffolding [62], Tree [13], Dynamic-join [60] etc. However, all of these algorithms are based on instantiation match. Clearly, instantiationless match suggests a radical departure from these instantiation-match algorithms. The work reported here suggests that a full instantiation match may not always be necessary, and that utilizing instantiationless match in such situations may potentially lead to substantial improvements in performance. Instantiationless match may be of particular relevance for production match algorithms in database environments [55], where space efficiency is of concern. Indeed, recently, Acharya and Tambe [1] have proposed algorithms based on instantiationless match for database environments. These algorithms, called collection-oriented algorithms, do not require a restriction on the expressiveness of the production system, and they do not change the worst case complexity of production match. However, because they can avoid instantiations under some circumstances, they can reduce the generation of cross products, and thus reduce the space and time consumed (by multiple orders of magnitude).

The principles for eliminating combinatorics identified in this article may also generalize beyond production match. For instance, the key idea behind instantiationless match is to avoid the explicit generation of cross-products. A similar idea has recently been used by Hubbe and Freuder [34] in their method, called *CPR*, for efficient representation of partial solutions in solving general constraint-satisfaction problems. CPR is used in conjunction with standard methods for solving constraint-satisfaction problems, such as backtracking. CPR avoids the creation of a complete cross product of values assigned to variables. Instead, it maintains the values that participate in the cross product in separate sets. Identifying the applicability of key principles such as this may result in new methods for attacking problems in combinatorial match and other research areas.

Acknowledgments

This research benefited from many interesting discussions with Allen Newell. Kevin Knight and the anonymous reviewers provided very helpful comments on the first draft of this paper.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD) under contract numbers F33615-87-C-1499 and N00039-86C-0033 (via subcontract from the Knowledge

Systems Laboratory, Stanford University), by the National Aeronautics and Space Administration under cooperative agreement number NCC 2-538.

References

1. A. Acharya and M. Tambe. Collection-oriented match: Scaling up the data in production systems. Tech. Rept. CMU-CS-92-218, School of Computer Science, Carnegie Mellon University, December, 1992.

2. P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. Proceedings of the National Conference on Artificial Intelligence, August, 1987, pp. 268-272.

3. B. P. Allen. Bringing up Soar in ART. Proceedings of the seventh Soar workshop.

4. E. Altmann. Toward human-scale task performance: A stage 4 learning system. School of Computer Science, Carnegie Mellon University. Thesis proposal.

5. J. R. Anderson. *The architecture of cognition.* Harvard University Press, Cambridge, Massachusetts, 1983.

6. F. Barachini. "The evolution of PAMELA". *Expert Systems 8*, 2 (1991), 87-98.

7. F. Barachini. Production match in Pamela. Private Communication.

8. F. Barachini and N. Theuretzbacher. The challenge of real-time process control for production systems. Proceedings of the National Conference on Artificial Intelligence, 1988, pp. 705-709.

9. F. Barachini and G. Verteneul. "Methodes de prediction du temps d'execution pour systemes experts temps reel". *Genie Logiciel et systemes et expertes 28* (September 1992).

10. F. Barachini, H. Mistelberger and A. Gupta. Run-time prediction for production systems. Proceedings of the Tenth National Conference on Artificial Intelligence, 1992, pp. 478-485.

11. A. Bayazitoglu, T. R. Johnson and J. W. Smith. A unique-attribute representation of annotated models that facilitates learning. Tech. Rept. OSU-LKBMS-92-100, Ohio State University Division of Medical Informatics, 1992.

12. A. Bayazitoglu, T. R. Johnson, and J. W. Smith. Limitations of the unique-attribute representation for a learning system. Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications (CAIA-93), 1993.

13. J. Bouaud. TREE: A heuristic driven join strategy of a rete-like pattern-matcher for a Soar-like production system. Tech. Rept. DIAM Rapport Interne RI-92-109, INSERM U194 Department intelligence Artificielle et Medecine, 1992.

14. J. Bouaud, and P. Zweigenbaum. A reconstruction of conceptual graphs on top of a production system. Proceedings of the 7th Annual Workshop on Conceptual Graphs, 1992.

15. L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming expert systems in OPS5: An introduction to rule-based programming.* Addison-Wesley, Reading, Massachusetts, 1985.

16. P. Chalasani, and E. Altmann. Comparing the representations in Soar and Theo. School of Computer Science, Carnegie Mellon University, Unpublished.

17. M. P. Chase, M. Zweben, R. L. Piazza, J. D. Burger, P. P. Maglio and H. Hirsh. Approximating learned search control knowledge. Proceedings of International Workshop on Machine Learning, 1989, pp. 218-220.

18. W. W. Cohen. Learning approximate control rules of high utility. Proceedings of the Sixth International Conference on Machine Learning, August, 1990, pp. 268-276.

19. K. Crawford and B. Kuipers. Negation and proof by contradiction in access-limited logic. Proceedings of the National Conference on Artificial Intelligence, August, 1991.

20. R. Dechter and J. Pearl. "Network-based heuristics for constraint-satisfaction problems". *Artificial Intelligence 34*, 1 (1988), 1-38.

21. G. F. DeJong and R. Mooney. "Explanation-based learning: An alternative view". *Machine Learning 1*, 2 (1986), 145-176.

22. R. Dodhiawala, N. S. Sridharan, P. Raulefs and C. Pickering. Real-Time AI systems: a definition and an architecture. Proceedings of the International Joint Conference on Artificial Intelligence, 1989, pp. 256-261.

23. O. Etzioni. *A structural theory of search control*. Ph.D. Th., School of Computer Science, Carnegie Mellon University, December 1990.

24. S. E. Fahlman. Three flavors of parallelism. Proceedings of Fourth National Conference of the Canadian Society for Computational Studies of Intelligence, May, 1982.

25. C. L. Forgy. "Rete: A fast algorithm for the many pattern/many object pattern match problem". *Artificial Intelligence 19*, 1 (1982), 17-37.

26. E. C. Freuder. Complexity of k-tree structured constraint-satisfaction problems. Proceedings of the Eighth National Conference on Artificial Intelligence, 1990, pp. 4-9.

27. M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, San Francisco, California, 1978.

28. A. Golding, P. S. Rosenbloom and J. E. Laird. Learning general search control from outside guidance. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, August, 1987, pp. 334-337.

29. D. N. Gordin and A. J. Pasik. Set-Oriented constructs: from Rete rule bases to database systems. Proceedings of the ACM SIGMOD Conference on Management of Data, 1991, pp. 60-67.

30. A. Gupta. *Parallelism in production systems*. Ph.D. Th., Computer Science Department, Carnegie Mellon University, 1986.

31. A. Gupta, M. Tambe, D. Kalp, C. L. Forgy and A. Newell. "Parallel implementation of OPS5 on the Encore multiprocessor: Results and analysis". *International Journal of Parallel Programming 17*, 2 (1988).

32. P. V. Haley. Real-time for Rete. Proceedings of ROBEXs'87: The Third Annual Workshop on Robotics and Expert Systems, 1987.

33. E. N. Hanson. Rule condition testing and action execution in Ariel. Proceedings of the ACM SIGMOD Conference on Management of Data, 1992, pp. 49-58.

34. P. D. Hubbe and E. C. Freuder. An efficient cross-product representation of the constraint-satisfaction problem search space. Proceedings of the National Conference on Artificial Intelligence, 1992.

35. G. A. Iba. "A heuristic approach to the discovery of macro-operators". *Machine Learning 3*, 4 (1989), 285-318.

36. Inference corp. ART-IM programming language reference manual. Reference manual.

37. T. Ishida. Optimizing rules in production system programs. Proceedings of the Seventh National Conference on Artifical Intelligence, 1988, pp. 699-704.

38. T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao, and J. Y. Read. "Real-time Knowledge-Based Systems". *AI magazine 9*, 1 (1988), 27-45.

39. J. E. Laird. *Soar user's manual: Version 4.0.* Intelligent Systems Laboratory, Palo Alto Research Center, Xerox Corporation, 1986. Reprinted and available from Carnegie Mellon University.

40. J. E. Laird, A. Newell and P. S. Rosenbloom. "Soar: An architecture for general intelligence". *Artificial Intelligence 33*, 1 (1987), 1-64.

41. J. E. Laird, P. S. Rosenbloom and A. Newell. "Chunking in Soar: The anatomy of a general learning mechanism". *Machine Learning 1*, 1 (1986), 11-46.

42. J. F. Lehman, R. L. Lewis and A. Newell. Natural language comprehension in Soar: Spring 1991. Tech. Rept. CMU-CS-91-117, School of Computer Science, Carnegie Mellon University, 1991.

43. X. Li, R. Krishnan and D. Steier. MFS: A study of model formulation within Soar. Tech. Repr Working paper 91-18, School of Urban and Public affairs, Carnegie Mellon University, 1991.

44. A. K. Mackworth. "Consistency in Networks of Relations". *Artificial Intelligence 8*, 1 (1977), 99-118.

45. A. K. Mackworth and E. C. Freuder. "The complexity of some polynomial network consistency algorithms for constraint satisfaction problems". *Artificial Intelligence 25*, 1 (1985), 65-74.

46. S. Markovitch and P. D. Scott. Information filters and their implementation in the SYLLOG system. Proceedings of the Sixth International Workshop on Machine Learning, 1989, pp. 404-407.

47. J. McDermott. "R1: A rule-based configurer of computer systems". *Artificial Intelligence 19*, 2 (1982), 39-88.

48. S. Minton. Selectively generalizing plans for problem-solving. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 1985, pp. 596-599.

49. S. Minton. Quantitative results concerning the utility of explanation-based learning. Proceedings of the Seventh National Conference on Artifical Intelligence, 1988, pp. 564-569.

50. S. Minton. *Learning Effective Search Control Knowledge: An explanation-based approach.* Ph.D. Th., Computer Science Department, Carnegie Mellon University, 1988.

51. S. Minton, J. G. Carbonell, C. A. Knoblock, D. Kuokka, O. Etzioni and Y. Gil. "Explanation-based learning: A problem solving perspective". *Machine Learning 40* (1989).

52. D. P. Miranker. *Treat: A new and efficient match algorithm for AI production systems.* Ph.D. Th., Computer Science Department, Columbia University, 1987.

53. D. P. Miranker. Treat: A better match algorithm for AI production systems. Proceedings of the Sixth National Conference on Artificial Intelligence, 1987, pp. 42-47.

54. D. P. Miranker. Ordering bindings in instantiationless match. Private Communication.

55. D. P. Miranker, D. A. Brant, B. Lofaso, and D. Gadbois. On the performance of lazy matching in production systems. Proceedings of the Eigth National Conference on Artificial Intelligence, 1990, pp. 685-692.

56. T. M. Mitchell. Becoming increasingly reactive. Proceedings of the Eighth National Conference on Artificial Intelligence, 1990, pp. 1051-1058.

57. T. M. Mitchell, R. M. Keller and S. T. Kedar-Cabelli. "Explanation-based generalization: A unifying view". *Machine Learning 1*, 1 (1986), 47-80.

58. P. Nayak, A. Gupta and P. S. Rosenbloom. Comparison of the Rete and Treat production matchers for Soar (A summary). Proceedings of the Seventh National Conference on Artificial Intelligence, 1988, pp. 693-698.

59. A. Newell. *Unified theories of cognition.* Harvard University Press, Cambridge, Massachusetts, 1990.

60. K. Oflazer. "Highly parallel execution of production systems: A model, algorithms and architecture". *New Generation Computing 10* (1992), 287-313.

61. M. Perlin. The match box algorithm for parallel production system match. Tech. Rept. CMU-CS-89-163, Computer Science Department, Carnegie Mellon University, 1989.

62. M. W. Perlin. Scaffolding the Rete network. Proceedings of the International Conference on Tools with Artificial Intelligence, November, 1990.

63. M. Perlin. Constraint satisfaction for production system match. Proceedings of the International Conference on Tools with Artificial Intelligence, 1992, pp. 28-35.

64. P. S. Rosenbloom and J. Aasman. Knowledge level and inductive uses of chunking (EBL). Proceedings of the National Conference on Artificial Intelligence, August, 1990, pp. 821-827.

65. P. S. Rosenbloom and J. E. Laird. Mapping explanation-based generalization onto Soar. Proceedings of the Fifth National Conference on Artificial Intelligence, 1986, pp. 561-567.

66. P. S. Rosenbloom, J. E. Laird, A. Newell and R. McCarl. "A preliminary analysis of the Soar architecture as a basis for general intelligence". *Artificial Intelligence 47*, 1-3 (1991), 289-325.

67. D. E. Smith and M. R. Genesereth. "Ordering conjunctive queries". *Artificial Intelligence 26* (1986), 171-215.

68. I. Stobie, M. Tambe and P. S. Rosenbloom. Flexible integration of path-planning capabilities. Proceedings of the SPIE conference on Mobile Robots, Novemeber, 1992.

69. S. Stolfo. "Initial performance of the DADO2 prototype". *IEEE Computer 20*, 1 (1987), 75-83.

70. M. Tambe. *Eliminating combinatorics from production match.* Ph.D. Th., School of Computer Science, Carnegie Mellon University, May 1991.

71. M. Tambe and P. S. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 1989, pp. 731-737.

72. M. Tambe, and P. S. Rosenbloom. A framework for investigating production system formulations with polynomially bounded match. Proceedings of the Eighth National Conference on Artificial Intelligence, 1990, pp. 693-400.

73. M. Tambe, D. Kalp, and P. S. Rosenbloom. An efficient match algorithm for production systems with linear-time match. Proceedings of the International Conference on Tools with Artificial Intelligence, 1992, pp. 36-43.

74. M. Tambe, D. Kalp, A. Gupta, C. L. Forgy, B. G. Milnes and A. Newell. Soar/PSM-E: Investigating match parallelism in a learning production system. Proceedings of the ACM/SIGPLAN Symposium on Parallel Programming: Experience with applications, languages, and systems, 1988, pp. 146-160.

75. M. Tambe, A. Newell and P. S. Rosenbloom. "The problem of expensive chunks and its solution by restricting expressiveness". *Machine Learning 5*, 3 (1990), 299-348.

76. C. Wang, A. K. Mok, and A. M. K. Cheng. MRL: a real-time rule-based production system. Proceedings of the Real-time Systems Symposium, December, 1990.

77. D. A. Waterman. *A guide to expert systems.* Addison-Wesley, Reading, Massachusetts, 1986.

78. D. A. Waterman and F. Hayes-Roth. *Pattern-directed inference systems.* Academic press, New York, New York, 1978.

79. R. C. Yee, S. Saxena, P. E. Utgoff and A. G. Barto. Explaining temporal differences to create useful concepts for evaluating states. Proceedings of the Eighth National Conference on Artificial Intelligence, 1990.

80. J. Yen. A principled approach to reasoning about specificity of rules. Proceedings of the Eighth National Conference on Artificial Intelligence, 1990, pp. 701-707.

Table of Contents

List of Figures

List of Tables